

# Índice

## Prefacio

- 1 - Los estudios. El Instituto de Informática
  - 2 - El equipamiento informático en los años setenta
  - 3 - El Método de trabajo en la década de los setenta
  - 4 - Los mainframes de IBM
  - 5 - El Lenguaje Cobol. El “Virus del Milenio”
  - 6 - El método de trabajo en la década de los ochenta
  - 7 – El Método de Programación Estructurada
  - 8 - El Sort (o el viejo problema de ordenar las cosas)
  - 9 - La irresistible irrupción del PC en la Empresa
  - 10 - El camino hacia las Bases de Datos Relacionales
  - 11 - La llegada de las Bases de Datos Relacionales
  - 12 - La Ventana Batch
  - 13 – Aparición fulgurante de las Metodologías de Desarrollo
  - 14 - Herramientas CASE hasta en la sopa
  - 15 - Los años noventa (I): La Guerra de los PC’s
  - 16 - Los años noventa (II): La ascensión de los minis
  - 17 - Cuando descubrimos el “procesamiento paralelo”
  - 18 - El Data Warehouse entró en nuestras vidas...
  - 19 - Y el Data Warehouse se convirtió en... Business Intelligence
  - 20 - El descubrimiento de la Minería... pero de Datos
  - 21 - La “burbuja puntocom” se hinchó... y explotó
- ## Epílogo

## Memorias de un Viejo Informático

*Macluskey, 2009-2014*

© Macluskey, 2014

Diseño de la portada: el autor.

Ilustración: Sala del Ordenador del *Mission Operations Center* de la NASA en la década de 1960 (dominio libre).

Todas las imágenes utilizadas en este volumen son:

o bien diseñadas y realizadas por el autor,

o bien de dominio libre,

o bien han sido publicadas bajo licencia Creative Commons,

en cuyo caso se especifica el hecho y la fuente de la que se ha obtenido la imagen.

Primera edición, Septiembre de 2014

*A Mari Carmen*, que me ha soportado mientras vivía todas las experiencias que  
narro aquí, y

*A Sara*, cuya infancia casi me pierdo convirtiéndome en alguien importante. Sin  
conseguirlo.

Ni falta que me hacía.

A ellas dos les debo todo.

Y, cómo no, a *Pedro Gómez Esteban*, de *Eltamiz.com*,  
que me prestó su maravilloso blog para que pudiera  
contar mi irrelevante historia a los cuatro vientos.

## Prefacio

Cuando Pedro Gómez-Esteban, el dueño y editor del blog [www.eltamiz.com](http://www.eltamiz.com), me pidió tiempo ha que escribiera algún artículo para publicar en *Elcedazo*, pensé que se había vuelto loco.

*Elcedazo* ([www.eltamiz.com/elcedazo](http://www.eltamiz.com/elcedazo)) es un blog comunitario que, dentro de *eltamiz* (uno de los mejores blogs de ciencia en español), ofrece un lugar idóneo para que personas que, como yo, no queremos complicarnos la vida creando y administrando un blog pudieran publicar artículos o trabajos interesantes que quisieran dar a conocer a la comunidad.

No creía yo que la historia, o mejor, las vivencias de un informático de los tiempos del cólera tuvieran mucho interés para los seguidores del blog, pero luego algunos otros asiduos también me pidieron lo mismo... así que por fin me decidí a escribir algunos artículos, no sólo contando mi irrelevante historia sino, además, divulgando parte de mis conocimientos y experiencias a lo largo de treinta y cinco años de profesión, cosa que sí me pareció que pudiera resultar interesante para algunos.

Y eso es lo que hice. Entre enero y julio de 2009 se publicaron veintitantas entradas de la “**Historia de un Viejo Informático**” en *elcedazo*, bastantes más de las que preveía al principio, y me ha sorprendido sinceramente la cálida acogida que los lectores tuvieron para tanta historia obsoleta. Así que seguí escribiendo más y más artículos...

...Y de esta manera hemos llegado hasta aquí.

Ésta es una recopilación, corregida y aumentada, pero sobre todo *corregida*, de los artículos publicados en *elcedazo*. Una parte de la estructura de la serie permanece en los capítulos, pero he decidido mantenerla así para facilitar, en su caso, la consulta del artículo correspondiente de la serie, dado que allí aparecen, además del texto, un montón de enlaces a páginas muy interesantes que, por motivos obvios, no es posible trasladar aquí.

Decidir el tono que deberían mantener los artículos me llevó alguna reflexión. Pensé que si usaba un tono académico y formal, teniendo en cuenta lo largos y densos que iban a ser todos y cada uno de los artículos, probablemente conseguiría que los lectores se aburrieran y dejaran la lectura.

Como mi objetivo era, por encima de todas las cosas, divulgar temas muy técnicos para los no iniciados, decidí que los artículos deberían mantener un tono cercano y desenfadado, divertido en ocasiones, que intentara precisamente dar una idea de lo divertida que es (o, al menos, ha sido) mi profesión y de lo bien que yo, personalmente, lo he pasado trabajando quince horas diarias y deseando volver al día siguiente a seguir trabajando un poco más... Y ese mismo tono desenfadado es el que he decidido mantener aquí, en realidad por las mismas razones: se pueden contar cosas muy interesantes de forma amena y sin necesidad de aburrir hasta a las ovejas... lo que no quiere decir que yo lo haya conseguido, claro.

Ahora bien, ese estilo desenfadado y lenguaraz no debe ocultar que **la información contenida en cada capítulo es, hasta donde he podido, rigurosa, cierta y, espero, de utilidad** para colegas que están empezando o atacando problemas que quizá otros hayan ya atacado.

No sólo anécdotas, chismorreos, curiosidades e historias varias encontraréis aquí, que también; hay mucha información sobre experiencias, métodos, técnicas y herramientas, muchas de ellas hoy en día obsoletas y prácticamente desconocidas, pero otras quizá igual de desconocidas, pero de una utilidad fuera de toda duda. Toda esta información que he ido atesorando con los años me gustaría que otros la pudieran aprovechar, una vez que se acerca el final de mi vida laboral y mi jubilación... ¡si es que el Gobierno de turno me deja jubilarme!

De las mucha tecnologías posibles que en la informática han sido desde mis comienzos, en los primeros años setenta, sólo hablaré aquí de aquellas en las que he estado personalmente involucrado en algún momento (o en muchos) de mi vida profesional... pero es que veréis que son bastantes.

Debido a una serie de circunstancias, entre las que no es la menor el que yo sea un *culo de mal asiento*, he tenido la oportunidad de estar presente en muchas de las movidas tecnológicas que han tenido lugar a lo largo de los años, así que igual hablaré de características de máquinas, como de métodos de desarrollo, herramientas software, problemática comercial... Pero no esperéis que hable en profundidad de algunas tecnologías, como AS/400, o Java, o Linux... no tengo conocimiento suficiente para poder hilvanar más de cuatro frases coherentes sobre cada una, que es incluso menos de lo que se puede encontrar en cualquier obra de referencia, como la Wikipedia, por ejemplo.

Aunque he buscado información en diversas fuentes para documentar de la mejor manera posible cada episodio, he de decir aquí que **la principal fuente de la que he sacado la información es mi memoria**, mi vieja, cansada y selectiva memoria, que, si hemos de creer a los neurólogos, con el transcurso del tiempo desecha, por inútil, lo superfluo y se queda sólo con lo que de verdad es importante... si en mi caso es así o no, lo podréis comprobar leyendo las páginas siguientes.

Algún lector bienintencionado solicitó en alguna ocasión que sería conveniente que diera referencias externas, quizá de la Wikipedia... pero es que en realidad estas Memorias de un Viejo Informático son una “fuente primaria”, casi como las memorias de un político o un deportista. En efecto, casi todo lo que aquí se puede leer ha sido extraído directamente de mi memoria o, en algún caso, de mis papeles amorosamente guardados a lo largo de los años, y como tal deben tomarse: como las memorias de un informático de los tiempos gloriosos, nada más.

En cualquier caso, nunca he tratado de escribir una “*Historia Oficial*” de nada. Es mi historia, y en ella prevalecen siempre mis recuerdos sobre cualquier otra información que haya podido encontrar en la Red o en cualquier otro sitio sobre los artilugios y acontecimientos que narran estas líneas; es posible que en ocasiones mis viejos recuerdos discrepen de lo normalmente aceptado, pero al fin y al cabo, repito, estoy escribiendo *mi* historia, tal como yo la recuerdo... y con la máxima honradez que me es posible. Eso es lo único que os garantizo: **Que contaré las cosas honradamente**, tal y como yo las viví en su momento y sin adornos florales de ningún tipo.

Pero tened indulgencia, mi vieja memoria va y viene... así que igual os cuento las mismas cosas en varios puntos diferentes del relato, mientras que otras más importantes me las dejaré en el tintero. ¡Qué se le va a hacer! Esto es lo que hay y, por más que me empeñe, nunca volveré a tener treinta años, así que pido perdón de antemano por los inevitables lapsus que cometeré.

Disfrutad, pues, de estas “**Memorias de un Viejo Informático**”. Verdaderamente, nada me haría más feliz que pasarais tan buenos ratos leyéndolas como los que he pasado yo escribiéndolas.

Madrid, 2009-2014

# 1 - Los estudios. El Instituto de Informática

**“¡Ingeniero Aeronáutico!”.**

Eso es lo que yo contestaba sin dudar cuando, en los primeros años sesenta del Siglo pasado, me preguntaban aquello tan manido de: “Y tú... ¿qué vas a ser de mayor, rico?”. Tenía yo siete u ocho años y no quería ser bombero, ni indio, ni siquiera americano, que era lo que por entonces querían ser todos mis amiguitos. Ya era yo raro por entonces, hijo de un humilde funcionario y de modista de barrio (casi igual que sucedía en las primeras temporadas de la serie de TV “Cuéntame”... había momentos en que parecía que estaba contando la historia de mi familia, no la de los Alcántara), queriendo ser Ingeniero y, además, para construir nada menos que aviones, cuando el mayor aeropuerto español de la época, Barajas, en Madrid, donde yo jamás había estado ni de visita, no haría más de doce o quince operaciones al día... así que imaginad la cara que ponía mi familia por entonces. E imaginad cómo de *especial* debo ser ahora, más de cincuenta años después.

Finalmente las circunstancias de la vida me llevaron a ser informático, cuando eso, en la década de los setenta, era casi tan raro como lo de ser Ingeniero Aeronáutico en la España de los sesenta.

Ésta es la historia de cómo acabé estudiando tan esotérica cosa, en vez Medicina, Derecho o Ciencias Exactas, como todo hijo de vecino...

En mi época escolar, tras los estudios de Primaria, que acababan a los diez años, se estudiaba un “Bachillerato Elemental” (de cuatro años, desde los once a los catorce años), que podía ser seguido de un “Bachillerato Superior” de dos años más (quince y dieciséis años). Lo recuerdo porque es fácil de recordar: Primero, 11 años; Segundo, 12; y así hasta Sexto, 16 años. Fácil. No como ahora...

El caso es que en Sexto de Bachillerato (Plan 1957), con dieciséis años, por tanto, y a principios de los setenta, había que decidir varias cosas importantes. Una: si iba a estudiar una Carrera Universitaria o me iba a poner a trabajar (cosa esta última que la precaria economía familiar agradecería bastante). Dos: en caso de elegir estudiar, qué Carrera elegir, para ir preparando el terreno, aunque esta decisión podría esperar hasta que aprobara el Preuniversitario (el famoso “Preu”, sustituido muy poco después por otro engendro similar: el COU) y, sobre todo, el temido examen final eliminatorio donde si aprobabas podías pasar a la Universidad y si suspendías... bueno, en mi caso, si suspendía, a trabajar de aprendiz o de botones en algún sitio cualquiera, claro.

Afortunadamente, aprobé. ¡Y con nota!

Decidí entonces, no sé muy bien por qué, que quería ser “*Especialista en Cerebros Electrónicos*”... o así.

Comenzaban a estar de moda por entonces los “Cerebros Electrónicos”, gracias, entre otras cosas, a las películas americanas en las que se podían ver a esos sofisticados engendros dominando el mundo... cuando lo que aparecía en pantalla era una unidad de cinta magnética toda llena de lucecitas, con la cinta girando *p’atrás* y *p’alante* todo el tiempo, como si la cabeza lectora no fuera capaz de leer correctamente la cinta.

Lo curioso del asunto es que esta práctica de poner una cinta magnética girando estúpidamente a un lado y a otro como imagen de *superpotentes ordenadores* persistió

durante muchos años en las pelis de acción o de ciencia ficción. Ciertamente, en los años sesenta y setenta del Siglo XX se sabía tan poco de “Cerebros Electrónicos” que ver una cinta moviéndose sola podía parecer el colmo de la sofisticación, pero ya en los noventa...

Además, todo el personal que salía en las pelis iba con bata blanca, así que debía ser una ocupación realmente importante... En una palabra, yo tampoco sabía nada sobre qué era la informática, pero quería hacer de ella mi profesión... ¡Qué inconsciente es la juventud! El caso es que comencé a informarme con los *MUY limitados medios* existentes en la época y lo que intuí, más que entendí, me gustó. Así que me reafirmé en mi opinión: ¡quería estudiar informática!... fuera eso lo que fuese.

La alternativa obvia de la época era estudiar en una Academia privada. Durante muchos años han funcionado este tipo de Academias, que daban, y supongo que siguen dando, una formación práctica, quizá poco académica pero suficiente para comenzar a trabajar programando o, lo que era más sencillo entonces, grabando datos, pues había mucho más trabajo en este área que programando. Pero había también una alternativa oficial para estudiar Informática. Bueno, *casi* oficial.

A finales de los sesenta se creó con dos sedes, una en Madrid y otra en San Sebastián, el **Instituto de Informática**. Algún año después se creó otra sede más en Barcelona. Era el *IDI* una cosa rara, con vocación universitaria pero sin serlo, que daba al final de los cinco años de estudios el Título (no oficial) de “*Técnico de Sistemas*”, sirviera este título para lo que fuera, que tampoco quedaba claro.

Para poder matricularse había que hacer un examen de acceso (además del de Preu o el de COU), tenía una matrícula más cara que la de la Universidad pero menos que estudiar informática en academias, y aseguraba (quiero decir que *sus responsables aseguraban*, no que estuviera asegurado) que, tarde o temprano, su titulación sería homologada a Título Universitario. Me presenté al examen de ingreso, sorprendentemente aprobé también y allí me planté en octubre de 1972 a comenzar los estudios de Informática... fuera eso lo que fuese.

Por cierto, el examen de ingreso era un lujo de modernidad: tipo test, marcabas a lápiz, para cada respuesta, una de las cuatro opciones disponibles en su casilla correspondiente, y luego la corrección la hacía... ¡**Un Ordenador!** leyendo las marcas ópticas de los exámenes: eso, en 1972, parecía algo extraído directamente del Proyecto Apolo.

La primera sorpresa agradable fue **el excelente nivel del profesorado**. En aquellos tiempos no había titulados oficiales en Informática, ni siquiera fuera de España, así que los únicos profesores disponibles eran profesionales que tuvieran experiencia en los temas que impartían y que les gustara enseñar y supongo que tener algún ingreso extra. La consecuencia es que *cada profesor enseñaba mayormente lo que sabía*. Quizá eso pudiera ser diferente de lo que sabía el profesor que daba la misma asignatura en la clase de al lado, pero lo que contaba era auténtico: eran los conocimientos que, como profesional, había adquirido trabajando en su empresa, que presumiblemente nos serían muy útiles a los alumnos unos años después.

Casi todos los profesores eran responsables de Departamentos de las principales compañías informáticas de la época: IBM, Univac, Bull, NCR..., o bien responsables del Departamento de Proceso de Datos de grandes empresas o instituciones que adoptaron tempranamente el uso de ordenadores en España: Ministerios, RENFE, Iberia, Telefónica... Para las empresas citadas era un honor que profesionales suyos fueran profesores en el IDI, así que no sólo no ponían pegas a su docencia (“poner pegas” es lo que, según me cuentan,



suele suceder en la actualidad), sino que alentaban esta actividad docente.

Los conocimientos que nos transmitían eran útiles de verdad. Y nos permitían incorporarnos al mercado laboral rápidamente, sin necesidad de grandes cursos adicionales... justo lo contrario que ocurre ahora, por cierto, pero ésa es otra historia y será contada en otro momento.

Un ejemplo: en Segundo de Carrera teníamos una asignatura de nombre “Informática Básica II”. El profesor que me tocó en suerte era Técnico de Sistemas en Univac, una de las compañías pioneras en computación. Univac se había fusionado con Sperry-Rand Corporation y se llamaba entonces Sperry-Univac; años después, en 1986, se fusionó con Burroughs Corp. para dar origen a Unisys, compañía que sigue funcionando en la actualidad.

El primer día de clase, nos dijo: “*Miren Vds., no sé muy bien qué tengo que contarles en esta asignatura, pero yo de lo que entiendo es del Univac 1110, así que les voy a contar cómo es y cómo funciona un Univac 1110. Y para que sepan Vds. cómo funcionan otros ordenadores, a mediados de curso vendrá un colega mío que les contará cómo es y cómo funciona el IBM 370...*”. Y eso hizo. Con pelos y señales.

El examen final consistió en escribir en ensamblador un programa de canal para el Univac 1110 que fuera capaz de leer un disco magnético con acceso directo... y lo hicimos, claro, y debimos de hacerlo bien, porque aprobamos la mayoría de alumnos.

En la siguiente ilustración, un incunable: los apuntes del primer día de clase de Segundo, en 1973, porque libros, lo que se dice libros, vimos muy poquitos; prácticamente toda la Carrera la estudiamos sobre apuntes. En 1973 *las páginas eran blancas*, lo prometo.



Seguro que algunos os estáis preguntando: ¿Cuál sería el currículum de semejante pseudo-Carrera (aunque a nosotros nos parecía una Carrera *completa*, sinceramente)? Pues en Primero de Carrera eran cuatro asignaturas solamente:

- **Informática Básica I** (donde te enseñaban qué era un ordenador, para qué servía, qué era un disco, una cinta... nadie tenía ni la menor idea de ordenadores al sentarse el primer día en clase).

- **Métodos Matemáticos I** (en realidad, Álgebra Lineal, pero bastante moderada comparada con lo el nivel que se daba en las Ingenierías).

- **Inglés I** (que empezaba por el “*I am, you are...*”. En España en aquellos tiempos el “Idioma Moderno” que se estudiaba en Colegios e Institutos era francés, *comment allez vous?*).

- **Programación I**, subdividida en tres sub-asignaturas a su vez, una por cada uno de los lenguajes de programación más importantes que había en la época: Fortran (para resolver problemas científicos), Cobol (para problemas comerciales) y Ensamblador, en

concreto el del UNIVAC 9200, una especie de clon del IBM 360.

Como veis, no se perdía el tiempo con tonterías: todo lo que se daba al principio era directamente utilizable para ponerte a trabajar al día siguiente de acabar Primero... y muchos lo hacían.

En Segundo había Informática Básica II, Métodos Matemáticos II (Cálculo, igual de moderado que el Álgebra de Primero), Inglés II (ahora ya tenías que saber al menos el *to be*), Programación II (Ensamblador más avanzado y Algol, un lenguaje precursor de Pascal y con menos éxito todavía), Física (Electricidad solamente, de la que casi no me acuerdo de nada) y Metodología (que en realidad era Lógica, la asignatura que más nos gustaba a todos: ¡era divertidísima!).

Quizá pudiera parecer que los estudios en aquel Instituto de Informática eran como los de una *SuperAcademia* de las que tanto proliferaban en la época... pues nada más lejos de la realidad. Es cierto que Primero y Segundo Curso eran eminentemente prácticos, siendo Informática Básica y los diversos lenguajes de Programación las asignaturas que más tiempo consumían (junto con Álgebra, en primero, y Cálculo, en segundo). Pero a partir de Tercero la orientación cambiaba: Estadística, Análisis Numérico y Teoría de la Información y la Codificación eran las asignaturas clave de Tercero (en ésta última ya se estudiaban los códigos de Huffman que utilizan todos los compresores de entonces y de ahora, por ejemplo), junto con Teleproceso, Algoritmos (Tecnología de la Programación era el nombre de esa asignatura) e Inteligencia Artificial, que tampoco es que hubiera mucho en este área en la época, pero se estudiaba lo que había.

En Cuarto, el *coco* era una asignatura preciosa (para mi gusto) de nombre “Técnicas de Optimización de Sistemas”, que tenía dos partes: Teoría de Colas (de sucesos estocásticos, distribuciones de Poisson, etc., imprescindible para poder configurar los Gestores de Teleproceso) y Programación Lineal y Dinámica (el Algoritmo *Simplex* o método de Dantzig, sobre todo). Además había más Teleproceso, más Inteligencia Artificial (Autómatas, Gramáticas y Lenguajes), etc.

Y en Quinto, por fin, más Teleproceso, Sistemas de Recuperación de Información, nombre esotérico para lo que ahora se llamaría “Bases de Datos”, Diseño de Compiladores, Reconocimiento de Formas y Teoría de Juegos, sobre todo, del Ajedrez, aunque el estudio de un juego tan complicado estaba literalmente en pañales en 1977: había la creencia general, que duró bastantes años, de que nunca jamás de los jamases un programa de ordenador podría llegar a vencer a un Gran Maestro, ni tan siquiera a ponerle a apuros... menos de veinte años después el Sistema de IBM *Deep Blue*, programado específicamente para la ocasión, venció no a un Gran Maestro cualquiera, sino al flamante Campeón Mundial de Ajedrez, uno de los mejores jugadores de la Historia en el culmen de su juego: Gary Kasparov... Ahora, prácticamente cualquier programa comercial pulverizaría a la gran mayoría de maestros y grandes maestros. Y es que *las ciencias adelantan que es una barbaridad*, como diría D. Hilarión en La Verbena de la Paloma, del maestro Tomás Bretón.

En definitiva, en el Instituto de Informática de los años 70 del siglo pasado **la preparación final que obteníamos nosotros, los alumnos, era extraordinaria**, en teoría y en la práctica, y nos habilitó para incorporarnos al mercado de trabajo de forma rápida y eficaz... ¡sobre todo para nuestros patronos!

Sin embargo, los medios materiales eran patéticamente escasos. Usábamos los ratitos libres en que pudiera utilizarse el ordenador del Ministerio de Educación, un Univac 9200 que era un *casi clónico* del IBM 360.

Las prácticas consistían en escribir un programa de cada lenguaje (Fortran, Cobol, Assembler, Algol...), que se llevaban a perforar (sin verificación, claro), compilar y, si por casualidad no tenía errores de compilación, ejecutar... que nunca se llegaba a ese paso, obviamente.

Estoy refiriéndome a **UN programa**.

*Al año.*

No había capacidad para más. ¡Igual que ahora, que a los alumnos los fríen a prácticas, casi todas completamente inútiles...! De cómo trabajábamos los informáticos de la época hablaré en los capítulos siguientes.

En 1974, una vez acabado Segundo fui llamado a filas. Sí, me fui a la *mili*. En aquella época el Servicio Militar era obligatorio, pero obligatorio-obligatorio. Había que estar preparados por si *los malos* nos invadían... Otra cosa es que nadie supiese ya, en 1974, quiénes eran esos malos. No había modo de librarse de la mili, salvo que tuvieras trescientas dioptrías o una pierna de madera. Ninguna de estas anomalías eran mi caso, lo mío era sólo pura torpeza, pero eso, como el valor, en la mili *se suponía*.

Por tanto, aunque ese año me matriculé en Tercero de Carrera, entre guardias, imaginarias y tiempos perdidos, la verdad es que no aprobé más que un par de asignaturas... y decidí terminar lo que quedaba de Tercero el año siguiente, una vez acabada la dichosa mili.

Influyó en la decisión de no comenzar un nuevo curso con no-sé-cuántas asignaturas pendientes del año anterior el hecho de que empezara a trabajar en otoño de ese mismo 1975, poco antes de la muerte del general Franco, como becario-programador en un gran banco nacional. De esto hablaré en el siguiente capítulo.

El caso es que las empresas nos buscaban, pues sabían de la calidad de la enseñanza del IDI comparada con la que se podía encontrar en los centros alternativos de enseñanza de informática, así que de hecho en Quinto ya todos estábamos trabajando... y esta circunstancia me recuerda una de las anécdotas más divertidas que vivimos esos años, que no me resisto a contaros ahora.

Un (excelente) profesor de una asignatura de nombre “Sistemas de Recuperación de Información” (ahora se llamaría “Bases de Datos” o algo parecido), estaba explicando el funcionamiento del IMS, la Base de Datos de IBM, que es la que él conocía... una compañera hizo una pregunta un tanto *extraña*, no sé cómo calificarla, teniendo en cuenta que todos estábamos ya trabajando en empresas reales. El buen hombre, extrañado, preguntó a mi compañera: “Pero... Señorita, ¿Vd. trabaja?” “Sí” “Ya, y ¿en qué trabaja Vd.?” “¡Construyo Alas Delta!”.

El pobre profesor se puso colorado como un tomate, dejó la tiza y se fue inmediatamente de clase... la carcajada se oyó desde el Rectorado. ¡Había ido a dar con la única persona de todo el Curso que no trabajaba en algo relacionado con nuestra profesión!, que ya es mala suerte.

Volviendo a mi Tercero de Carrera, ese curso (1975-76) fue *curioso* en la historia del Instituto de Informática, porque por fin se cambió a su sede “definitiva” (ahora está en *otra* “sede definitiva”, por cierto). Había estado ubicado en la Calle Vitrubio, en pleno centro de Madrid, justo al lado del Paseo de la Castellana, compartiendo instalaciones con un organismo muy importante denominado “Banco Mundial”, que ni sabía entonces ni sé ahora muy bien a qué se dedica.

Con la cada vez mayor afluencia de estudiantes de esa cosa nueva de los *cerebros electrónicos*, el edificio de la Calle Vitrubio se estaba quedando pequeño a pasos

agigantados. Así que se pusieron manos a la obra y prepararon la mudanza.

Pero esto es España, o mejor aún, aquello era la España de los años setenta, y resulta que hubo que dejar las instalaciones antiguas antes de que estuviesen preparadas las nuevas... y el curso 1974-75 fuimos como el Holandés Errante. Cada curso repartido por diferentes ubicaciones y Escuelas; donde se encontró alguna aula libre, allí que se mandó a algún grupo. Hubo gente en las Escuelas de Ingeniería de Caminos, de Obras Públicas, de Montes, Forestales, Industriales... Sin orden ni concierto, los de Tercero estábamos en dos o tres sitios diferentes, y lo mismo pasaba con el resto de cursos.

Al fin, en octubre de 1975, se entregaron las nuevas instalaciones del Campus de Vallecas, a mitad de camino entre Vallecas-Villa y Santa Eugenia, o sea, en pleno extrarradio madrileño, a una hora de autobús del centro de la ciudad, autobús que pasaba cada treinta o cuarenta minutos, así que más te valía no perder el que te venía bien, que además te dejaba a quince minutos de barrizal de tu edificio...

Sin embargo, esto seguía siendo España, así que las instalaciones se entregaron... ¡sin electricidad! O sea, no había luz eléctrica. De veras. Las clases de la mañana se impartían con luz natural, poca o mucha, que estábamos en invierno, pero yo, que tenía mi curso por la tarde y las clases se impartían de seis a once de la noche, no empecé las clases hasta enero, cuando por fin pudieron conectar la electricidad...

Y no, igual que ahora, en 1975 los ordenadores, sin electricidad, no funcionaban, pero en realidad eso no era ningún problema: **no había ordenadores de ningún tipo** en el Instituto de Informática.

Bueno, sorprendentemente aprobé lo que me quedaba de Tercero ese año, y Cuarto el siguiente y Quinto al otro... y me gradué en 1978, con el título de “**Licenciado en Informática**”. Sí, “Licenciado”, no “Ingeniero”, enseguida os cuento la razón. En 1977 el Instituto de Informática se convirtió por fin, huelgas de estudiantes y manifestaciones diversas incluidas, en la **Facultad de Informática**, adscrita a la Universidad Politécnica de Madrid donde, por cierto, el resto de Centros eran todos Escuelas Técnicas Superiores, y los títulos impartidos, Ingenieros. Por lo tanto, fue por entonces cuando la Informática entró por la puerta grande en el mundo universitario. Bueno, quizá no tan grande. Mi promoción, que se licenció en 1978, fue la primera que terminó efectivamente como graduados universitarios, aunque, naturalmente, a los “Técnicos de Sistemas” graduados en años anteriores se les convalidó el título a “Licenciados”.

En resumidas cuentas, durante bastantes años los informáticos universitarios que se graduaban en España lo hacían siendo Licenciados, no Ingenieros como ahora. Las razones últimas de por qué sucedió esto que se rumoreaban por entonces (la razón real no la sé y no conozco a nadie que la conozca) eran dos: por una parte, el supuesto temor de los Ingenieros a tener que compartir no sé qué status con una caterva de neo-Ingenieros indocumentados en *esa cosa tan rara* nueva, que ni siquiera producían algo tangible, como un buen puente, un barco mercante o un avión.

Y por la otra, Don José García Santesmases, Doctor en Ciencias Físicas y principal valedor y estandarte de la incipiente informática española, no era ingeniero y, por lo tanto, podría tener dificultades en obtener una Cátedra o Rectoría o algo así en una Escuela Técnica para Ingenieros. Visto con la perspectiva de los años parece algo estúpido, lo sé, pero se oía con mucha fuerza en aquellos años. Por cierto, no busquéis mucha información sobre D. José... la primera noticia que aparece en Google si le buscas es su necrológica, de octubre de 1989.

La verdad es que a nosotros, los informáticos de la época, nos traía al paio ser una

cosa u otra. Nos daba exactamente lo mismo. Como ya dije, todos los estudiantes de Quinto (y el noventa por ciento de los de Cuarto y quizá el sesenta o setenta de Tercero) trabajábamos ya como informáticos en diferentes empresas o instituciones, con sueldos razonables, incluso buenos, y excelentes expectativas de trabajo en un momento de tremenda convulsión social en España: tasas de inflación superiores al 30% anual, fuerte ajuste económico tras la caída de la Dictadura y la crisis del petróleo de los años 70, paro elevado, alta tensión social, terrorismo, serio riesgo de involución (fijaos en los sucesos del 23-F de 1981, sólo unos pocos años después), etc. En una palabra, toda España era un hervidero durante aquellos convulsos años.

Recordad que la votación de la Ley de Reforma Democrática, que abrió el melón de la democracia en España, tuvo lugar en junio de 1977 y la Constitución la votamos en diciembre de 1978, cuando yo ya era Licenciado y llevaba más de tres años trabajando.

De cómo eran los ordenadores de la época, sus características, cómo se trabajaba con ellos (en tantas ocasiones, *a pesar de ellos*) y demás zarandajas hablaré en el siguiente capítulo.

## **2 - El equipamiento informático en los años setenta**

Tras la descripción de cómo eran los estudios de Informática en los tiempos del cólera voy a intentar describir ahora cómo era la “Informática de verdad” en las empresas aquellos años, es decir, qué me encontré por ahí fuera en mis comienzos en el mundo laboral.

Porque, efectivamente, en 1975 comencé a trabajar de programador en un gran Banco. Me ficharon como Becario, forma de contratación que ya era normal en la época, aunque al acabar el año inicial de beca me hicieron un contrato fijo (en la época no existía el contrato temporal, al menos no como ahora). Mi primer sueldo fue de casi veinte mil pesetas netas (unos 120 Euros) al mes... por catorce pagas. Una fortuna. De hecho, mi primer sueldo era casi igual que el de mi padre en aquellos tiempos, a pesar de sus diez o doce trienios, que en el funcionariado representaban una parte muy importante de la paga. ¡Y ni siquiera había acabado la Carrera, que aún estaba en Tercero!

El equipamiento del Banco era realmente impresionante: Dos ordenadores NCR Century 200 con nada menos que 32 KB de memoria cada uno, dos unidades de disco cada uno, tres unidades de cinta magnética (dos, de 1600 bpi, y la otra, de 800 bpi, que se utilizaba para entrada de datos), lector de tarjetas perforadas y una impresora realmente rápida cada ordenador. Además, la instalación tenía un switch que permitía asignar discos o cintas físicamente conectados a un ordenador al otro, por lo que en caso de necesidad un ordenador podía correr un programa con seis cintas magnéticas y cuatro discos simultáneamente. Que yo sepa, nunca tuvimos tal necesidad, pero en fin.

Ahora voy a hacer un poco de arqueología informática y describir con algún detalle los elementos más importantes que componían a la bestia. No es nada sencillo encontrar documentación, fotos o diagramas de estos viejos ordenadores en la red. De hecho, es difícilísimo encontrar nada interesante de ordenadores de la gama Century, es como si incluso la propia NCR, que sigue existiendo en la actualidad, se hubiera olvidado de ellos.

Veamos antes qué pinta tenían estos sistemas:

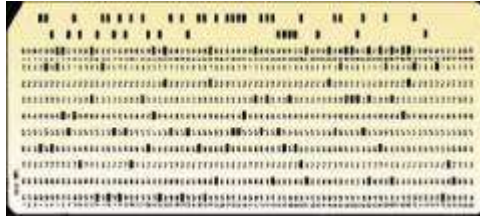


**La CPU.** El NCR Century 200 era una evolución del Century 100, con mayor potencia de proceso y mayor capacidad de memoria. No sé si aparecía en algún sitio de la documentación técnica cuántos megahercios tenía, que en realidad sería algún ciento escaso de kilohercios, ni cuantas instrucciones por segundo era capaz de ejecutar, que no serían más allá de algunas decenas de miles. Por ejemplo, las instrucciones de multiplicar y dividir, aunque tenían su propia instrucción ensamblador, se ejecutaban por software, por un procedimiento muy análogo al que usamos los humanos para multiplicar o dividir... los que se acuerden, claro. Su lenguaje ensamblador era bastante potente, de hecho más que el de su principal competencia de la época, el Sistema IBM 360.

**Lenguaje de Programación.** Su lenguaje nativo era NEAT/3, específico de NCR: una curiosa mezcla entre Ensamblador y Cobol, con los inconvenientes de ambos y pocas de sus ventajas, según mi opinión personal; sin embargo había fervientes defensores del NEAT/3 como el mejor lenguaje inventado jamás. Como siempre, para gustos hay colores. En realidad era una evolución del ensamblador y, santo y seña de NCR, durante mucho tiempo fue el único lenguaje que admitía esta máquina. Lo mejor que tenía el NEAT/3 eran sus potentes instrucciones para usar tablas internas, que no tenían parangón en ningún otro lenguaje.

Sin embargo, a principios de los setenta NCR implementó por fin Cobol para la gama Century y yo, de hecho, apenas programé nada en NEAT/3, pues lo hice casi desde el principio en Cobol. Dado que, pese a estar hoy en día denostado y a que no lo enseñe casi nadie, es en Cobol en lo que están programados la mayoría de Sistemas de Información que gobiernan el mundo, le dedicaré un capítulo aparte más adelante. Se lo merece.

**Entrada Primaria del Sistema.** El dispositivo de entrada del Sistema era el lector de tarjetas perforadas. Los programas los leía en fichas perforadas, el Boot (el arranque de la máquina) lo hacía leyendo las instrucciones de tarjetas perforadas y los trabajos del sistema también los leía de tarjetas perforadas. Esto es una tarjeta perforada:

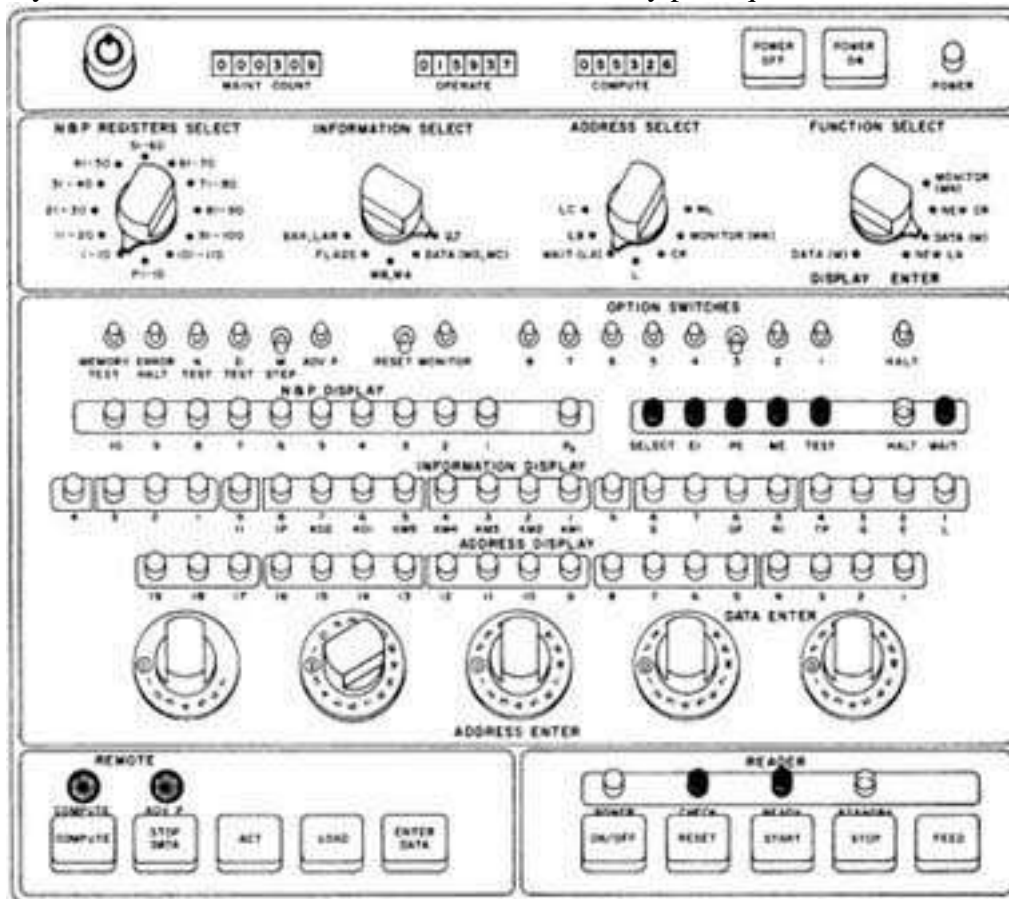


Más adelante contaré cómo era el proceso de arrancar la máquina, que era fascinante, muy distinto de lo que ahora ocurre.

**Consola del Sistema.** Un panel con diales y lucecitas que se encendían y apagaban en función de la dirección en que se ejecutaban las instrucciones o su contenido, así como diversas luces para comunicar incidencias. A diferencia de las consolas de lucecitas actuales, que sólo tienen utilidad, y poca, para los técnicos de mantenimiento, la consola se operaba habitualmente para diversas funciones y había que ser capaz de interpretar la información que el sistema comunicaba encendiendo y apagando sus luces.

Así, si las luces de contenido estaban en la situación: *Apagada-Apagada-Encendida-Encendida-Apagada-Apagada-Apagada-Encendida*, esto quería decir “00110001”, es decir, un “31” en Hexadecimal, o sea, “1” en ASCII, o “49” en binario, etc.

En la siguiente ilustración siguiente podéis admirar cómo era esa preciosa consola del Century 200, con sus lucecitas, sus botones, sus diales y palanquitas...





Para la comunicación “sofisticada” se disponía de un teletipo. Así, como suena. Un vulgar y lento teletipo. El nombre técnico del artilugio era “I/O Writer”, lo que da una idea exacta de lo que era: una especie de máquina de escribir con un teclado *qwerty* reducido y un rollo de papel pijama para que el ordenador respondiera escribiendo en él.

El Operador daba la orden (por ejemplo NEXTDO NOMINAS, es decir: ejecuta ahora el programa “NOMINAS”), orden que aparecía impresa en el papel pijama conforme la ibas escribiendo, y aceptabas la instrucción pulsando “Control-Bell”, que era la letra G pulsando CTRL simultáneamente. Se llamaba “Control-Bell” porque al pulsarla sonaba una campanita, ¡*clinnn!*!, para marcar la aceptación.

Entonces, si el sistema tenía que dar información de lo que fuera, lo hacía escribiendo en el papel pautado, letra a letra a un lentísimo ritmo y usando códigos para casi todo. Por ejemplo, podía contestar “NOMINAS AF”, lo que significaba que no existía ningún programa que se llamara NOMINAS, o “PEPITO 0B”, lo que significaba que no había espacio para ubicar el fichero PEPITO en el disco. Y así todo.

**Memoria.** Como dije, 32 KB. O sea, 32.768 bytes, ni uno más. Un KB, un kilobyte, en definitiva, no tiene 1000 bytes, como cualquiera podría suponer. En realidad un kilobyte tiene 1024 bytes, que es el resultado de elevar 2 a la décima potencia. Cosas de los informáticos... Esta memoria era de ferritas.

Una curiosidad: el Emblema de los Licenciados e Ingenieros de Informática, creado por Real Orden de no sé qué día de 1975, es precisamente una ferrita con sus cuatro cables atravesándola y unas ramas de laurel, olivo y tal; muy pocos saben hoy en día qué demonios es esa “rosquilla con rayas entremedias”, pero gracias a ellas estamos hoy donde estamos, pues la memoria principal de todos los ordenadores de los años 60 y 70 era de ferritas.



Las ferritas estaban en un armario enorme atestado literalmente de cables. Eran con mucho el elemento más caro del sistema, pues las ferritas se enhebraban una a una, a mano, y en 32 KB había 262.144 ferritas, que son muchas, enhebradas cada una por cuatro cables: más de un millón de ellos. Al año más o menos de estar yo trabajando, el Banco se gastó una fortuna (creo recordar que unos veinte millones de pesetas de la época en cada ordenador), para ampliar la capacidad de sus dos ordenadores de 32 a 64 KB. Poca cosa, ¿no?

Pero es que con este cambio, se pudo actualizar el Sistema Operativo también, pasando del *B1* original que funcionaba con las 32 KB, al novísimo *B3*. ¡Y el *B3* permitía la multiprogramación! Es decir, con 32 KB el ordenador sólo podía ejecutar un programa al tiempo, mientras que, con 64, se podían hacer **dos** particiones de 32 KB y correr **dos**

programas simultáneamente, reduciendo casi a la mitad el tiempo total necesario para ejecutar los diversos procesos. Tengo que decir aquí que el Sistema Operativo ocupaba en memoria menos de 5 KB, con lo que tenías nada menos que 27 KB y pico para tu programa. Eso era entonces... ¡Una barbaridad de espacio!

**Discos Magnéticos.** Eran “removibles”, es decir, en cada unidad de disco se ponía uno u otro disco según la necesidad. Estos discos removibles eran cilíndricos, del tamaño de una tartera grande (unos treinta y cinco centímetros de diámetro y unos quince de alto) y la increíble capacidad de algo más de 4 MB. Sí, sí, no me he equivocado, he dicho 4 MB, o sea, cuatro millones de caracteres. Y pico.

En la ilustración, tomada de la Wikipedia, una unidad de disco del IBM 360, muy similar a las de NCR de la época.



Obsérvese el disco, de cinco o seis platos y del tamaño aproximado de un LP de vinilo, la muesca para poder abrir el receptáculo donde va el disco y los botones de control, a la izquierda.

La unidad de disco era como una lavadora de carga superior (de hecho, más grande que una lavadora), con una tapa que se levantaba hacia arriba y permitía introducir o extraer el disco mediante un ingenioso dispositivo adaptado en la funda, de tal forma que el delicado contenido del disco *rara vez* sufría daños.

Una vez introducido el disco en la *lavadora* (todos la llamábamos así), le dabas al botón de “Start” y esperabas sus buenos tres o cuatro minutos para que alcanzara la velocidad de rotación requerida y se pusiera “Ready” encendiéndose el inevitable pilotito verde, momento en el que el disco estaba operativo.

Para extraer el disco dabas al botón de “Stop”, rojo, como no podía ser de otro modo, y esperabas sus otros dos o tres minutos a que se parara para poder abrir la unidad y extraerlo. Podéis imaginar que, con semejante capacidad, los discos se reservaban sólo para el software (los programas propiamente dichos) y algunos ficheros de datos fijos como Sucursales, Condiciones de Liquidación, Fechas y así.

Los ficheros Maestros (Clientes, Cuentas, Empleados, etc.) y los de Movimientos eran, por supuesto, bastante más grandes que eso, así que no cabían en los discos. Entonces seguro que os estaréis preguntando: Pero... ¿dónde se guardaban los ficheros importantes (bueno, y los menos importantes también)? Pues, naturalmente, en...

**Cintas Magnéticas.** Armarios enormes con dos carretes, uno a la izquierda y otro a

la derecha (había modelos de otros fabricantes que tenían los carretes arriba y abajo) y un sofisticado mecanismo de lectura.



El operador colocaba en el carrete de la izquierda la bobina de cinta magnética con el fichero a leer o escribir, extraía, desenrollando la bobina, el comienzo de la cinta y, pasándola a través del mecanismo de lectura, la enrollaba en el carrete de la derecha.

Este mecanismo de lectura o escritura funcionaba a base de hacer un vacío controlado tanto antes como después de la cabeza lectora/grabadora, para evitar tirones que podrían romper la cinta, que al fin y al cabo era de plástico, similar a las de las cintas en cassette aunque más ancha y más gruesa.

Después se pulsaba el inevitable botón de “Start”, se hacía el vacío y la unidad buscaba el comienzo legible de la cinta magnética. Este “comienzo legible” era en realidad tan sólo una pequeña tira, de medio centímetro o así, de material reflectante. La unidad de cinta tenía un emisor de luz justo antes de la cabeza lectora/grabadora, así como una célula fotoeléctrica que detectaba el destello de la tira reflectante al pasar y reflejar la luz; ése era el comienzo útil de la cinta, y similar sistema indicaba el fin de la cinta antes de que quedara desbobinada de su carrete... lo que de todos modos sucedía de vez en cuando.

Una vez encontrado el principio, la unidad quedaba en “Ready” hasta que el programa diera la instrucción de lectura o escritura pertinente. Y sí, el indicador de “Ready” era verde.



En una cinta de 2400 pies como la de la IBM 729 de la ilustración (Wikipedia, TheSentinel64, CC by SA 2.5), que era el tamaño normal de las cintas de entonces, con sus 28 centímetros de diámetro, cabrían alrededor de treinta o cuarenta millones de caracteres (unos 40 MB, vaya), por lo que los ficheros más grandes, como el de Cuentas, Clientes, etc., no cabían enteros en una sola cinta, sino que necesitaban varias, denominadas “secciones”, que había que montar en orden... y a mano. En ocasiones el Operador se equivocaba y montaba la sección tres antes que la dos. Ya podéis imaginar el desaguisado que se montaba.

Naturalmente, mientras la unidad de cinta rebobinaba la sección para permitir su cambio, el programa (o sea, todo el ordenador) se quedaba tranquilamente parado, esperando que el operador informara por el teletipo que la cinta se había cambiado para continuar el proceso.

Las cintas magnéticas eran la base de aquella informática del achelense: prácticamente todos los procesos leían y escribían la información importante en cintas magnéticas. Las unidades de cinta generalmente estaban todas ellas juntitas, ocho, diez, veinte... dependiendo del tamaño del Centro de Cálculo, colocadas en batería y cercanas a una pared, no pegadas, puesto que para arreglarlas cuando se rompían o para hacer mantenimiento había que acceder a la parte trasera: quitando el panel quedaban al descubierto las tripas de la unidad, toda llena de tubos y cables y motores...

Una de las anécdotas más curiosas que conozco que tenga que ver con cintas magnéticas me la refirió un colega, compañero de la Universidad, que esos años trabajaba precisamente en mantenimiento de equipos informáticos. Contaba este hombre que recibieron un aviso de un gran cliente institucional (un ministerio, o Hacienda, o algo así), porque de pronto todas las cintas magnéticas que estaban funcionando normalmente (es decir, unas leyendo, otras escribiendo, etc., en total como diez cintas), todas las cintas, en

fin, se pusieron simultáneamente a rebobinar su carrete, independientemente del punto en que se encontraran, arruinando, por tanto, todos los trabajos que estaban haciendo, que tuvieron que ser rearrancados, con la consiguiente pérdida de tiempo y mosqueo del personal.

Mi colega fue rápidamente a ver qué había pasado y, tras hacer todos los tests y pruebas pertinentes, no encontró nada: todas las cintas estaban en perfecto estado. Así que anotaron el incidente dentro de la categoría de “*poltergeist*”, es decir, ni idea de qué ha pasado, y lo dejaron correr. Pero el caso es que dos o tres días después, ocurrió lo mismo, y además más o menos a la misma hora: las cinco de la tarde.

Los virus informáticos no habían sido inventados todavía, así que no eran sospechosos... Nueva excursión, nueva comprobación y todas las unidades de cinta seguían estando perfectas. La incidencia fue elevada a la categoría de “*poltergeist al cuadrado*”.

Al día siguiente, la misma historia, con el mismo resultado. Todo el mundo se comenzó a poner muy nervioso, así que mi amigo decidió que todos los días a las cinco de la tarde (como los toros) estaría allí como un clavo para ver *in situ* qué rayos pasaba. El primer día, nada. El segundo, igual: nada.

Pero el tercero... de pronto, a las cinco y diez en punto, todas las unidades de cinta, todas a la vez, comenzaron a rebobinar y expulsar la cinta, igualito que antes. Revisó lo que había en máquina, otra vez las unidades... ¡Nada!

El cuarto día volvió a ocurrir, esta vez a las cinco y siete minutos... pero ese día mi colega, un tipo avisado, sí se dio cuenta de algo raro: justo antes del evento había entrevisto algo, como un fogonazo, un breve flash, y a continuación todas las cintas comenzaron a rebobinar...

La investigación sobre el caso de *las-cintas-que-rebobinan* se convirtió entonces casi en un capítulo de Colombo... y cuento ya la solución final al enigma: En aquellos tiempos, fines de los setenta y principios de los ochenta, un Centro de Cálculo moderno todo lleno de aparatos impresionantes y llenos de lucecitas operado por gente muy seria con bata era un argumento más de marketing. Cuando una compañía instalaba uno de estos intentaba que se supiera, así que era muy común instalarlo con grandes ventanales a la calle, para que los viandantes pudieran observar lo importantes y modernos que eran en esa compañía o en ese ministerio, tanto da.

El caso es que ese Centro de Cálculo no era una excepción y tenía toda una pared de cristal con vistas a la calle y, sobre todo, con vistas de la calle *hacia adentro*. Las unidades de cinta estaban justo en la zona contraria al ventanal, para que se vieran bien, pues eran lo más vistoso de todo. Y por la calle circulaban vehículos, claro, incluso autobuses. ¿Qué ocurría? Entre cinco y cinco y diez, dependiendo del tráfico, pasaba por la calle un autobús urbano que giraba justo enfrente del Centro de Cálculo. A esa hora y en esa época del año el sol estaba situado de tal modo que incidía en las ventanas del autobús al girar y su reflejo atravesaba toda la Sala a la velocidad de la luz para incidir directamente justo en las cabezas lectoras de las unidades de cinta.

¿Qué interpretaban esas cabezas lectoras, todas ellas? ¿Que habían alcanzado el fin de cinta!, es decir, que habían detectado el destello que producía el pedacito de material reflectante que indica precisamente eso. La cabeza no sabía si el destello era consecuencia del reflejo de la luz interna en el indicador de fin de cinta, o producido por un autobús urbano girando... Y hacían lo que tenían programado en ese caso: rebobinar la cinta y bajar la mampara de cristal para dar acceso a la cinta y que el operador pueda cambiarla.

Para resolver el problema de *las-cintas-que-rebobinaban*, por tanto, bastó con poner

unas humildes persianas en el ventanal. Pero pocos años después todo esto cambió, sobre todo a partir del atentado de ETA contra las instalaciones de Telefónica en su Centro de Datos de la Calle Ríos Rosas de Madrid en abril de 1982, donde el objetivo fueron precisamente los ordenadores que controlaban el tráfico de datos, lo que realmente causó un gran trastorno a las empresas que usaban estos servicios. Fue entonces cuando los Centros de Cálculo fueron escondidos en mazmorras bajo siete llaves y protegidos por bunkers más o menos seguros: todos se dieron cuenta de que los ordenadores no sólo eran un elemento de marketing, sino que comenzaban a ser estratégicos para las operaciones de las empresas o instituciones.

**Impresora.** La joya de la corona de NCR. Una impresora de líneas de última generación que imprimía en *papel pijama* de 132 caracteres, y lo hacía rapidísimo. Era de metro veinte o metro treinta de altura, alrededor de un metro y medio de ancho y uno de profundo.

Tenía un rodillo de 132 caracteres de ancho con todos los caracteres representados. Es decir, en cada posición, y a lo largo de la circunferencia del rodillo, estaban todos los caracteres imprimibles, los de entonces, quiero decir: las mayúsculas, los números y unos veinte o veinticinco caracteres especiales, como el punto, la coma, el asterisco, los paréntesis, etc. Cuando mirabas al rodillo de frente, veías una fila de 132 *Aes*, debajo una de 132 *Bes*, luego de *Ces*, y así sucesivamente.

En el otro lado, y al otro lado del papel, una vez colocado correctamente, estaban los martillos.

Si tenía que escribir una F, el martillo esperaba a que pasara la F del rodillo justo en el punto de golpeo y entonces saltaba y golpeaba al papel, que quedaba marcado con la letra F. Si tenía que imprimir un espacio... el martillo no golpeaba y dejaba el carácter correspondiente en blanco, ahorrándose así una fila de caracteres en el rodillo.

Aquí podéis observar un detalle de uno de estos rodillos, en concreto con la línea de caracteres correspondiente al carácter del tanto por ciento, “%”.



Cuando todos los caracteres de la línea habían sido impresos, la impresora saltaba a la línea siguiente, y así sucesivamente. Para que el carácter quedara impreso hacía falta tinta... que se obtenía de un papel de calco que se colocaba entre el rodillo y el papel: artesanal, sí, pero funcionaba. Y a una velocidad muy alta, pues era capaz de imprimir 1.500 líneas por minuto. La coordinación mecánica necesaria para realizar este proceso correctamente era muy notable para la época. A cambio, era el dispositivo que más se estropeaba, fácilmente una o dos veces por semana.

Una de las gracias de aquella impresora es que tenía duplicados los diez dígitos del 0 al 9 en dos partes opuestas del rodillo, por lo que si las líneas a imprimir sólo tenían dígitos, algo muy típico en la banca (balances de cuentas, extractos, apuntes contables, etc.), la impresora era capaz de alcanzar el doble de velocidad que cuando también había letras, es decir, llegar a unas 3.000 líneas por minuto. Una barbaridad, incluso para nuestros tiempos. Y, eso sí, hacía ruido. *Mucho ruido*. Las Salas de Ordenador de la época eran unos sitios realmente ruidosos...

Y *fríos*, por cierto. Yo creo que los operadores iban siempre con bata, encima del traje, no para no ponerse perdidos de tinta, que también, sino para no pelarse de frío. La mayoría de programadores y analistas teníamos siempre un jersey en un cajón para ponernos encima si teníamos que bajar a la Sala del Ordenador, sobre todo en verano.

**SopORTE.** Gratuito. Sí, es cierto: *gratuito*.

El coste del ordenador era tan alto, su mantenimiento tan costoso y los salarios, en comparación, tan ridículos, que las compañías de ordenadores te “regalaban” uno o varios Técnicos para arreglar lo que se fuera estropeando, tanto en el hardware como en el software.

Si tu ordenador era realmente grande los Técnicos *vivían* en tu instalación, para no perder ni siquiera el tiempo en desplazarse a arreglar el problema. En aquellos tiempos la instalación del Banco donde yo trabajaba no era demasiado grande (aunque el banco sí lo fuera, al menos con los criterios de la época), así que los técnicos sólo iban cuando se rompía algo... o sea, casi todos los días.

En una palabra, el hardware era caro, muy caro, y los servicios profesionales, comparativamente, eran baratos, así que cuando algo se rompía, se arreglaba: los técnicos de mantenimiento, armados de polímetro y soldador, averiguaban dónde estaba el error, sustituían exclusivamente el transistor dañado, o el condensador, o lo que fuera, y a seguir funcionando... y esto dio origen a multitud de anécdotas.

Recuerdo especialmente una que le ocurrió al técnico de mantenimiento de los equipos de grabación del Banco...

Parece ser que una determinada grabadora (la *máquina de grabar*, no la persona que la usaba) tenía errores intermitentes e insertaba un espacio en cualquier lugar, arruinando el registro grabado: posiciones de cuentas corrientes, movimientos de valores... toda la operativa del banco se grababa en cinta para su proceso por el ordenador central, gracias a sus imponentes 32KB de memoria... y a nuestros maravillosos programas, desde luego.

El técnico revisó la máquina de arriba abajo, la probó y no encontró nada raro: funcionaba perfectamente. Al día siguiente, nuevo aviso: la máquina seguía insertando espacios cuando le venía en gana... nueva revisión, mismos resultados... y así varias veces. Como el técnico en cuestión era un tipo muy dicharachero y con todos hablaba, en todo Proceso de Datos nos enteramos de lo que le estaba pasando al buen hombre con la dichosa grabadora.

Al final el técnico decidió *irse a vivir* a la Sala de Grabación (un lugar ciertamente agradable para nosotros, los chicos, todo lleno de Señoras y Señoritas de buen ver) para intentar determinar cuál podía ser el problema con la dichosa máquina. Entonces esto era normal, ahora parecería una extravagancia.

Al cabo de un rato observó que la grabadora (la *persona*, no la máquina) que estaba grabando datos en aquel momento, una señora de edad mediana y busto prominente, por decirlo de forma amable, se aupó para revisar los papeles que tenía ligeramente lejos de su alcance, y como consecuencia del movimiento, uno de sus senos se apoyó gentilmente... justamente encima de la barra espaciadora.

Entonces nos enteramos todos (ya digo, el problema que había con la-grabadora-que-metía-un-espacio-de-vez-en-cuando había trascendido ya de lo que era el Departamento de Grabación y lo conocíamos todos en Proceso de Datos) de que para solucionar el problema, bastaba con asignar a esa grabadora (*máquina*) concreta a otra grabadora (*persona*) menos *dotada físicamente*... y el *sucedido* nos sirvió para echarnos



unas risas durante mucho, pero mucho tiempo... incluso aún sonrío cuando lo recuerdo.

**El Arranque del Sistema (el Boot).** Lo de llamarlo “*Boot*” viene de “BootStrap” (ver el artículo de la Wikipedia inglesa sobre “*Boot*ing” si tenéis interés en saber de dónde viene tan curioso nombrecito), y es lo mismo que el IPL (“Initial Program Load”). De la españolización de “boot” viene lo de “*Botar la máquina*” que aún se escucha a veces.

Para arrancar aquel monstruo de ordenador se hacían las operaciones que contaré a continuación, porque no bastaba con apretar el botón y esperar un poco, como ahora, no...

El Century-200 tenía un Sistema Operativo tipo “DOS”, es decir, “Disk Operating System”, lo que quería decir que el Sistema estaba en un disco: el “Disco del Sistema”, naturalmente. Recordad que los discos tenían algo más de cuatro megas, así que podéis haceros una idea de la enorme complejidad del Sistema Operativo: todo él cabía en cuatro megas y aún sobraban dos y media para que las usaras para otras cosas, por ejemplo, tus programas.

El Sistema Operativo estaba efectivamente en disco, pero no existían cosas como “BIOS”, “ROM Memory”, etc. O sea, una vez encendida la máquina no había nada en memoria, ni forma de cargar nada automáticamente. Básicamente, tenías *un pedazo de hierro*. Encendido, sí, y caro, también, pero un hierro, al fin. Había que cargar el Sistema de forma manual. Usando, cómo no, un taco de fichas perforadas (no más de ocho o diez, recuerdo), pues fichas perforadas era lo único que el ordenador era capaz de leer por hardware.

El procedimiento era el siguiente:

**1** En primer lugar, se tomaba el bloque de tarjetas perforadas de arranque del Sistema y se colocaba en la lectora de fichas (“entrada primaria del sistema”, así se llamaba de forma oficial la lectora de tarjetas). El ordenador estaba parado, es decir, el switch de “Halt” estaba activado: cuando se activaba de forma manual, el ordenador interrumpía su operación y se quedaba “clavado” en la instrucción que estuviera ejecutando en ese momento y así se quedaba hasta que bajaras la palanquita y le dieras a la tecla de “Compute”.

**2** El Operador señalaba en el dial la dirección de carga del programa (*era la 00A0, qué cosas más raras se recuerdan al cabo de los años...*). Entonces pulsaba la tecla “Load” de la consola. Esto hacía que, por hardware, se leyera la primera tarjeta del taco y se cargara en la dirección marcada (en la 00A0, claro).

**3** A continuación (con la misma dirección 00A0 marcada), pulsaba la tecla “Act”. Esta tecla llenaba el registro de instrucciones, marcando que la próxima instrucción a ejecutar fuera la 00A0, la que marcaba el dial. Es decir, apuntando al principio del contenido de esa primera tarjeta que se acababa de cargar en la memoria. Esta tarjeta (todas las del taco, en realidad) estaba perforada en “multipunch”, es decir, tenía más agujeros de los que el código Hollerith permitía para las tarjetas perforadas, pues representaba caracteres binarios, con las instrucciones, en código máquina puro, que debía ejecutar el ordenador. O sea, el taco de tarjetas del Boot no era más que... ¡una BIOS de cartón!

**4** El Operador bajaba el switch de “Halt” y pulsaba la tecla “Compute”. Esto hacía que se comenzara a ejecutar el programa contenido en la dirección marcada. El contenido de los ochenta caracteres de esta primera tarjeta tenía el código suficiente como para poder leer el resto de fichas del taco, cargarlas en las direcciones sucesivas (la primera en la 00F0, la segunda en la 0140, y así) y, al detectar el fin del taco, ceder control a esta nueva pieza de código. En esos quinientos o seiscientos bytes cabía todo el código necesario para ir al



disco, recuperar el Sistema, cargarlo en la máquina, arrancar los procesos pertinentes, etc. Compacto, ¿no?

5 Cuando terminaba todos estos procesos, el teletipo de la consola emitía un “READY” (con su campanita al final y en mayúsculas, que no existían las minúsculas), y el ordenador estaba listo para aceptar trabajos. Todo el proceso podía tardar uno o dos minutos.

Otra anécdota de aquellos tiempos: recuerdo cierta vez que estuvimos dos días con uno de los dos ordenadores parado porque no había manera de que arrancara. Los técnicos de NCR estuvieron horas y horas revisando todos los componentes como ya dije, con soldadores, polímetros y cosas así: lo de los diagnósticos automáticos ni siquiera imaginábamos que *pudiera ser inventado* algún día.

...Al final resultó que una de las fichas del taco del IPL se había deteriorado, tenía un agujero que no debía tener (vaya Vd. a saber por qué) y naturalmente la instrucción que cargaba era errónea, por lo que al llegar allí y no tener el Sistema plenamente cargado, el proceso de arranque se quedaba tieso, sin decir ni mu. Y todos locos buscando día y noche. En fin. Cosas que pasaban en la Prehistoria...

Cuando un programa “cascaba” (o sea, daba un error fatal, un zapatazo, un abend, terminaba de mala manera, lo que ahora sería un pantallazo azul) la única posibilidad de averiguar qué había pasado era emitir un volcado de memoria, un “dump” en hexadecimal, con todo el contenido de la memoria del ordenador, que se imprimía en papel y te enviaban para averiguar el motivo del “casque”... y casi siempre con prisa, podéis imaginar.

Con tu programa a un lado y el volcado de memoria al otro, y perpetrado con varios lápices o bolis de colores, empezabas a marcar el contenido de cada campo, pintabas rayas de conexión, veías en qué instrucción había fallado y así... hasta que, unos minutos o unas horas más tarde, por fin sabías el motivo del error, para corregirlo.

Todos nosotros sumábamos y restábamos en hexadecimal casi mejor que en decimal. ¿He dicho ya que no había calculadoras que operaran en hexadecimal? Bueno, ni casi en decimal: eran unos trastos eléctricos grandotes que imprimían sus cálculos en una tira de papel... aún hoy se pueden ver calculadoras de éstas en algunas oficinas.

Ya podéis suponer que la preocupación por salvar árboles no se había puesto aún de moda: consumíamos cantidades ingentes de papel, pero porque tampoco había otra alternativa: toda la comunicación máquina-hombre se hacía en papel (pijama). Otros sistemas tenían ya sus pantallas antediluvianas (aunque para los operadores exclusivamente); NCR en los 70, no.

Quizá estéis pensando que qué se podría hacer con un ordenador con tan poquísima potencia...

Pues casi nada: llevar **toda** la información del Banco. Todas las aplicaciones importantes estaban escritas y en perfecto funcionamiento: Clientes, Cuentas Personales, Créditos y Préstamos, Liquidación de Cuentas, Depósitos, Contabilidad, Nómina, Cartera de Efectos, Valores con sus decenas de sub-aplicaciones, como Bolsa, Renta Fija, Ampliaciones de Capital, Abono de Cupón, Custodia, Traspasos, etc., en fin, todas las aplicaciones de la época estaban informatizadas y todas las comunicaciones al cliente se imprimían ya por las fastuosas impresoras que os he contado.

Por fin, todo el software era hecho ex profeso para el Banco: eso de comprar

software de otros no existía entonces, salvo los propios programas del Sistema Operativo, claro, que venían *de serie* con el ordenador.

Obviamente, para sacar partido a estos “mastodontes” (por el tamaño, desde luego, puesto que obviamente no por su capacidad, aunque entonces no pensábamos precisamente eso de ellos), es decir, para escribir las aplicaciones que funcionarían en ellos, se seguía un método de trabajo *ligeramente* distinto del que se usa ahora para escribir software de aplicación...

### 3 - El Método de trabajo en la década de los setenta

Como comentaba en el capítulo anterior, el mercado de software apenas existía en los años 70 y 80 del siglo pasado. El software básico (Sistema Operativo, Compilador, Sort, etc.) se compraba al fabricante de la máquina, había algún software de propósito general de fabricantes independientes (Bases de Datos, algún programa de gestión de ficheros y poco más) y casi ningún Software de Aplicación (lo que ahora se llama pomposamente un “ERP”, por *Enterprise Resource Planning*, es decir, la Contabilidad, la Nómina y otras por el estilo).

En una palabra, en los años 70 y primeros 80 del siglo pasado todos los programas de gestión de las empresas estaban hechos ex profeso **para** las propias empresas y, además, casi siempre por personal **de** las propias empresas.

Muy pocas empresas se dedicaban a ofrecer servicios profesionales en subcontratación, y las que lo hacían era para incorporar al analista o programador al equipo del cliente y trabajar codo con codo con ellos durante una temporada, generalmente larga, lo que después se llamó “bodyshopping”, qué nombre más horroroso... pero qué bien define la actividad, pardiez. O sea, casi no se contrataban proyectos “llave en mano”, que es lo habitual en nuestros tiempos.

Eso quería decir que los Departamentos de *Proceso de Datos* de las empresas o instituciones tenían personal propio y dedicado que escribía los programas para resolver las necesidades de esas empresas o instituciones. “*Proceso de Datos*” era el nombre que casi todas las empresas usaban en la época; con el tiempo los nombres se fueron adornando, complicando y cambiando según las modas del momento.

De acuerdo, pero... ¿**Cómo se trabajaba en el Pleistoceno de la Informática?**

Generalmente, aunque no siempre, había una persona (“*Analista Funcional Jefe de Proyecto*” era su denominación más habitual) que se ocupaba de la comunicación con los Usuarios, como Contabilidad, Cuentas Corrientes, Préstamos, etc. Esta figura, que casi siempre era “*un figura*”, se encargaba de recibir peticiones del Usuario, proponer soluciones, parir Sistemas de Información y dirigir al equipo de programadores que debían programarlos.

Una vez consensuada la Aplicación realizaba el **Análisis Funcional** (no, no es el mismo “Análisis Funcional” que se estudia en matemáticas, sino la descripción más o menos ordenada de las funciones que debe cubrir el Sistema de Información a diseñar).

Solíanse utilizar principalmente dos medios para realizar tarea tan importante, a saber: **Uno**: la servilleta del bar (si estaba manchada de mayonesa, mejor).



**Dos**: la comunicación verbal, en una reunión, donde contaba lo que se suponía que había que hacer y donde algún avanzado hasta tomaba notas y todo.

Os preguntaráis quizá que quiénes eran esos “*Analistas Funcionales Jefes de*

*Proyecto” y programadores que trabajaban ya en el Banco* cuando yo entré allí... porque yo era de una de las primeras promociones de Informática y aún estaba en Tercero de Carrera.

Pues se trataba de personal del propio Banco, de las áreas administrativas (cajeros, auxiliares, botones, etc), que fue seleccionado mediante un examen para ser formado específicamente en el ordenador que hubiera adquirido el Banco, en este caso NCR, durante varios meses.

Salvo excepciones, no eran universitarios (en la España de los sesenta y setenta no lo era casi nadie), pero hicieron estupendamente su trabajo: las aplicaciones más importantes (Cuentas Corrientes, Depósitos, Valores, etc) las diseñaron y escribieron ellos solitos... y funcionaban. Y, sobre todo, ¡eran unos artistas *emborronando servilletas*!

Años después, en muchos casos la vida fue injusta con esos grandes profesionales. La “*titulitis*” que apareció en España en los años ochenta y noventa los dejó fuera; muchos de ellos están “prejubilados” o simplemente “despedidos” de mala manera. Becario, jovenzuelo e inexperto, y sin saber una palabra de banca, ellos me acogieron, me tutelaron y me enseñaron todo lo que sabían. Aprendí muchísimo de estos *profesionales como la copa de un pino*. Quiero expresar desde estas humildes líneas mi Admiración y Respeto por ellos y hacerles el homenaje que, que yo sepa, nadie les ha hecho. **¡Gracias, amigos!**

El siguiente paso era repartir el trabajo. En realidad tampoco había mucho que repartir: lo normal es que con cada “Analista Funcional Jefe de Proyecto” hubiera uno o quizá dos programadores... ¡o tres! que atendían al área usuaria, así que no era difícil en realidad decidir quién iba a encargarse de qué.

La plantilla de Análisis y Programación de una gran empresa de la época podía componerse de **entre treinta y cincuenta personas**. Teniendo en cuenta que todas las Aplicaciones importantes funcionaban y que se daba servicio a las nuevas necesidades en un tiempo razonable, la productividad era muy alta, desde luego.

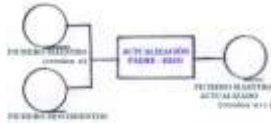
No perdíamos mucho tiempo en documentar, ni tampoco en hacer interminables memorandos justificando el coste: nos limitábamos a escribir el software, ponerlo en Producción, formar a los usuarios en su uso y dar soporte después. En definitiva, era para eso para lo que nos pagaban nuestros jugosos salarios las empresas: para escribir software y ponerlo en Producción, no para emborronar papeles... En estos tiempos modernos da muchas veces la sensación de que el objetivo de los informáticos es justamente lo contrario.

En fin, se ve que entonces éramos todos unos vagos, a los que no nos gustaba nada escribir... aunque cualquiera lo diría, viendo los ladrillos que escribo, aunque he de reconocer que a mí siempre me ha gustado darle a la pluma, o a las teclas, en su defecto.

No existían las Bases de Datos. Bueno, sí que existían, pero con capacidades muy escasas todavía y, que yo sepa, no había casi ninguna adecuada para los Sistemas de NCR. Sólo “Total”, de Cincom Systems, tuvo disponible a finales de los setenta una versión de Base de Datos para la gama Century de NCR. En el Banco se hizo entonces una prueba con esa Base de Datos, pero los resultados fueron bastante lamentables. Quizá hiciera falta mucha más memoria para que fuera bien, pero al precio que tenía la memoria... Se descartó, vaya.

Por consiguiente, los ficheros del Banco, los importantes, donde estaba toda la información sensible eran... eso: ficheros. Secuenciales.

Prácticamente todos ellos residían en Cinta Magnética, en una o en varias secciones, según su tamaño. Para realizar la puesta al día de estos ficheros, se ejecutaba un proceso de actualización conocido como “*Padre-Hijo*”, cuyo diagrama básico tenéis a continuación.



Para actualizar el fichero maestro se montaba en una unidad de cinta la versión actual de ese Fichero Maestro, en otro armario, una cinta virgen (o usada, puesto que se reutilizaban una y otra vez) para acoger la versión siguiente del mismo Fichero y se ejecutaba un programa de actualización (que solían ser los más complicados de la instalación) que, teniendo en cuenta el fichero de movimientos (o *los ficheros* de movimientos, pues en ocasiones se precisaba tener más de uno, y que por lo general estaban también en otra cinta magnética), generaba la versión actualizada de dicho Fichero Maestro.

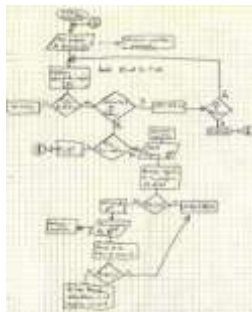
Se guardaban varias versiones de cada Fichero (entre 4 y 8, según lo importante que fuera) y se guardaban también los ficheros de movimientos de esos días para poder repetir un proceso que hubiera resultado erróneo por cualquier causa, generalmente porque el programa no hacía bien su trabajo... aunque en ocasiones (raras, la verdad) el Operador se equivocaba y montaba las cintas al revés, la de entrada donde debía estar la de salida y viceversa, y se machacaba la ultima versión del Fichero de Clientes o el que fuera...

Pero no pasaba nada: siempre recuperamos los procesos y nunca, nunca, perdimos información, estoy seguro. Bueno, *casi* seguro.

El programador, que sabía **qué** programa tenía que hacer, incluso, casi siempre, **lo que** tenía que hacer el programa, se ponía a ello. El primer paso era siempre “*hacer el mono*”. O sea, el organigrama. Los finos lo llamaban *ordinograma*, para distinguirlo del Organigrama de la Empresa, aunque éste último no hacía falta: todos sabíamos quién era quién esos años, no como ahora, que casi ni sabemos quién es nuestro jefe... ni mucho menos *por qué* es nuestro jefe.

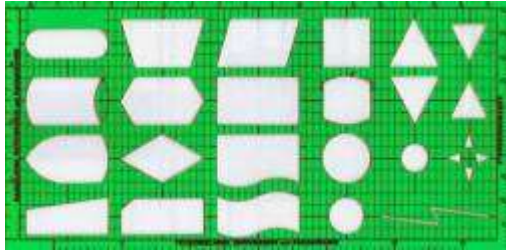
“Hacer el mono” era en realidad pintar en un papel (no, en *muchos* papeles, en realidad) el diagrama de secuencia del programa. En *código spaghetti*, claro... era el único que había por entonces, tanto que ni siquiera sabíamos que ése era su nombre.

En la imagen podéis admirar un Organigrama típico de aquellos años: la lectura de un registro determinado en un fichero indexado.



En el caso de que fueras un tipo limpito (no como el guarrete que pintó el Organigrama que aparece arriba, o sea, *ejem*, yo), entonces, en lugar de hacer tu organigrama a mano alzada utilizabas una plantilla, que entonces era un componente básico del kit de supervivencia del informático, para pintar convenientemente las diferentes instrucciones, bifurcaciones, etc.

Aquí podéis admirar una de esas plantillas:



En cualquier caso, los saltos de un lado a otro del flujo según las condiciones que se fueran satisfaciendo se pintaban con flechitas, que tendían a entremezclarse y volar de arriba abajo, de derecha a izquierda y, en realidad, en todas direcciones... de ahí la denominación de “*código spaghetti*” para los programas resultantes.

Este término (lo de *código spaghetti*) es claramente peyorativo, según bien dice la Wikipedia, pero es que *fue inventado muchos años después*. En los años setenta ésa era **la** manera de programar. **La única**, no había otra. Hasta 1983 u 1984 no empezó a imponerse eso de la “Programación Estructurada”, pero aún hubo que mantener programas antiguos, puro spaghetti, durante muchos, muchos años.

Una vez hecho el organigrama más o menos detallado, según la experiencia y habilidad del programador, el programador comenzaba a codificarlo, es decir, a traducirlo al lenguaje que el ordenador entiende (NEAT/3, Cobol, Fortran o lo que fuera). Para ello agarraba un taco de Hojas de Codificación en blanco y comenzaba a escribir allí su magnífico código.

En la imagen siguiente tenéis, por ejemplo, la fotografía de una típica Hoja de Codificación Cobol.



Ciertos programadores más inteligentes que la media tenían (*teníamos*, claro está) fotocopias de las hojas de codificación donde casi siempre era todo igual: nombre del programa, tipo de máquina, autor, etc., lo que nos permitía, rellenando las escasas partes que cambiaban, ahorrarnos escribir un poco. Pero poco, no os creáis...

Casi todos usábamos lápiz para poder corregir los errores (el portaminas de 0,5 era el rey, yo me acuerdo de mi precioso *Cross* plateado que alguien me birló, *sniff*) y solíamos tener simultáneamente seis o siete montones de hojas con las diferentes partes del programa que se iban codificando a la vez... y todo ello, bajo la continua consulta al organigrama... ¿He dicho ya que entonces no había pantallas, ni editores de texto, ni nada parecido?

Al cabo de varios días, dependiendo de la complejidad del programa y tu habilidad, terminabas el trabajo de codificación. La complejidad media de los programas que hacíamos entonces era bastante alta comparada con la de los que se hacen ahora. Preferíamos hacer un único programa muy complicado que tres más sencillos... si seguís leyendo os daréis cuenta cabal de por qué.

Con tu programa terminado, enviabas el taco de hojas al Departamento de Grabación, para que las perforaran. O sea, que cada línea que tú habías escrito fuera

copiada literalmente a una ficha perforada, con las perforaciones equivalentes al código que tú habías parido según el código Hollerith, así llamado por el inventor de la tarjeta perforada, Herman Hollerith.

En el *Departamento de Grabación*, donde una pléyade de grabadoras o perforistas (y lo digo en femenino porque, en todos los sitios en que he estado, en Grabación había exclusivamente mujeres, no me preguntéis por qué) se ocupaban de recibir documentación de Oficinas, Servicios Centrales, etc. y grabarlas en cinta o tarjeta perforada, según el programa que leyera los datos.

Para hacerlo se usaban máquinas perforadoras, como la de la imagen, una IBM 029 igualita a la del Banco con la que yo me pegaba cada lunes y cada martes...



Tus hojas de codificación, con palabras extraños y nombres impronunciables, como debe ser, se ponían en la cola hasta que le tocaba su turno.

...Y tardaba. Siempre tardaba, primero porque, lógicamente, tenía prioridad el trabajo diario (la contabilidad debía cerrar cada día, porque si no...) y segundo, porque a las perforistas no les gustaba nada, pero nada-nada, perforar programas: cientos o miles de fichas perforadas por programa, un trabajo muy feo, y lo digo por experiencia porque, cuando un programa te corría prisa de verdad, te ibas a Grabación y aprovechabas que alguna perforadora estuviera libre para perforarte tú mismo el programa. Y era un trabajo feo de verdad.

¿He dicho ya que a los Programadores no se nos caían los anillos si teníamos que perforar, grabar u operar en la máquina, cuando hacía falta? Y a los Analistas, tampoco, por cierto.

Además, para asegurar la máxima calidad en la grabación, todo el trabajo se *verificaba*. Esto consistía en volver a poner el taco de tarjetas grabadas en el alimentador de la perforadora, ponerla en “*modo verificación*” y volver a teclear de nuevo todo el programa. Ahora la máquina no perforaba, sino que se aseguraba de que lo que tecleabas coincidía con lo ya perforado. Si no era así, avisaba y había que ver dónde estaba el error, o en la tarjeta o en lo que habías tú tecleado en la verificación, y corregirlo.

Un tostón enorme y, encima, caro, por lo que en ocasiones no se verificaban los programas... y luego se producían muchos más errores.

Bien, han pasado unos días más y por fin te devuelven un taco de quinientas o mil o dos mil fichas con tu programa perforadito y atado con una goma elástica. Más te vale que las gomas con que el taco viene sujeto no se rompan: si se cae al suelo y se desordena... te puedes pasar un par de días hasta recolocararlo todo de nuevo. Las fichas perforadas nuevas (o sea, para perforar) venían en paquetes de 2.500 fichas. Cada paquete medía unos

cincuenta centímetros de largo por unos veinte de ancho y diez de alto y pesaba lo menos diez o doce kilos.

O sea, un programa de 1.000 fichas, bastante normal en la época, ocupaba un espacio de unos veinte centímetros, pesaba alrededor de cinco kilos y tardaba en ser leído por la lectora de fichas tres o cuatro minutos... si no se atascaba durante la lectura.

¿He dicho ya que, para ser un buen informático, además de tener una mente lúcida, también había que poseer unos buenos bíceps?

Entonces enviabas al Ordenador tu taco de fichas, para compilar el programa. “*Compilación*” es el proceso por el que el ordenador lee tu programa, verifica que cumple la sintaxis del lenguaje de que se trate, que los campos que has utilizado están correctamente definidos, etc...

Si la compilación acaba con “cero errores”, es decir, cumple todas las normas y es *aparentemente* un programa de verdad, y no el Quijote ni la lista de la compra, entonces el compilador traduce tus instrucciones en Cobol o el lenguaje que sea al código máquina que de verdad entiende el ordenador, que sólo sabe de ceros y unos, recordad.

Una compilación típica en esa época solía durar entre diez y quince minutos, con el ordenador utilizado en exclusiva para compilar. Como consecuencia, tu “MOVE ZERO TO CAMPO”, se convertía en una instrucción máquina parecida a, digamos, “64002E4C0E002540”, que sería la traducción en código máquina de tu estupenda MOVE.

Pues bien, repito, mandabas tu programa a compilar a la Sala del Ordenador... y vuelta a la espera.

El ordenador era monotarea, es decir, hace una sola cosa a la vez. Si el Operador lo dedica a compilar, no hace otra cosa. Y si está pendiente de ejecutar el Balance de Contabilidad, o el Abono de Cupón del Banco Hispano Americano, o la Liquidación de Intereses de los Depósitos a Plazo, pues no se compila nada, como es lógico. Las compilaciones tenían normalmente una prioridad... digamos, *baja*.

Algún año después el ordenador fue ya multitarea (en realidad, *bitarea*), por lo que la espera media se redujo de dos o tres días a sólo uno o dos. Así que, si te corría prisa que se compilara, podías hacer dos cosas:

**Una:** Irte un domingo al Banco y, mientras el único Operador que iba los domingos trasteaba en uno de los ordenadores, tú encendías el otro y te lo apropiabas durante unas horas. Yo esto lo hice muchas veces, y sin cobrar horas extras, ¿eh?... Claro que ahora tampoco se cobran, así que no os extrañará nada. Casi podríamos decir que, durante esas horas, era mi *ordenador personal*, ¿no?

**Dos:** Revisarte de arriba abajo tu taco gigante de fichas para detectar errores de perforación, o tuyos de programación. Y esto tenía guasa. Pero guasa. Te podías pasar un día entero para encontrar tres o cuatro errores (o treinta o cuarenta, si no habían verificado), que podías arreglar previamente a la primera compilación, eliminando, con suerte, una pasada (y una espera, en consecuencia).

Y así, por fin, los hados se habían conjurado con Isis y Zoroastro y tu programa había sido compilado...

... con el resultado de que normalmente tenía unos pocos, o muchos, errores de compilación: sintaxis, malas perforaciones, tontadas tuyas... en fin, lo de siempre.

Recibías entonces el resultado de la compilación: el listado en papel pijama, donde el compilador había marcado gentilmente para ti las líneas erróneas, eso sí, en ocasiones de forma tan críptica que no tenías ni idea de qué era lo que le pasaba a la dichosa línea.





Entonces ibas a por más hojas de codificación e ibas reescribiendo las líneas erróneas.

También podías seleccionar esas fichas erróneas en tu taco de fichas (no podéis imaginar la habilidad de *tahúr* que desarrollamos manejando la baraja de fichas perforadas: ningún amigo quería jugar al poker con nosotros, por si acaso) y marcar allí mismo los errores, sobre todo si eras tú mismo quien iba a perforar las correcciones: cuando no eran muchas, generalmente bajabas directamente a perforarlas, para así ganar algo de tiempo.

Una vez solventados los errores o, al menos, *eso es lo que tú te creías*, recomenzaba el ciclo: envío a perforar, recibir las nuevas fichas, recolocarlas si era preciso en el programa original, enviarlo nuevamente a compilar...

Tras varios intentos, dependiendo de lo torpe que fueras, por fin ha llegado tu programa sin errores de compilación. Pues ahora hay que probarlo. Es decir, asegurarse de que el programa, de verdad, hace lo que *se supone que debe hacer*, y no sólo lo que tú, programador, *te crees que debe hacer*, o peor, lo que *en realidad has programado*, que quizá no tiene nada que ver con lo que querías programar...

Para ello no queda más remedio que ejecutar el programa con los ficheros que luego va a utilizar de verdad o unos similares de prueba, aunque la verdad es que entonces nadie tenía ficheros de prueba de nada: se usaban los ficheros reales, y punto. Aún no había ninguna “Ley de Protección de Datos” y nosotros éramos gente muy seria.

Entonces solicitabas la prueba a la Sala del Ordenador, rellenando un papel al efecto especificando los ficheros que necesitaba tu programa, la salida esperada y si había que contestar algo por consola: muchos programas preguntaban cosas, por ejemplo, “Diga Vd. la fecha de proceso”, y el operador debía contestar en la propia consola la fecha pedida, que tu programa debía validar para que fuera correcta... bueno, deber, deber... debía, pero no siempre lo hacía.

Podéis imaginar que *también había que esperar para probar*, claro. Por ello, era común hacer una “Prueba en Mesa” antes de probar, incluso podías hacerla durante el proceso de puesta a cero errores de compilación. Por cierto, de todo esto en la Red no hay nada, y la verdad es que no me extraña...

Esta Prueba en Mesa significaba que *tú ibas a ser el ordenador* y a ejecutar tu programa instrucción a instrucción, para detectar fallos antes de la costosa prueba de verdad en máquina.

Te inventabas o tomabas de la realidad unos datos de entrada y pacientemente te ponías a desentrañar tu spaghetti, anotando: aquí leo el fichero A (y anotabas el contenido del siguiente registro del fichero A en su sitio), ahora comparo si F es mayor que G. Si es mayor, salto a Z (y mirabas si lo que tenías apuntado en F era mayor que G; si era así

seguías por Z, si no, continuabas), ibas machacando los valores de los campos con los nuevos valores... Y así durante horas.

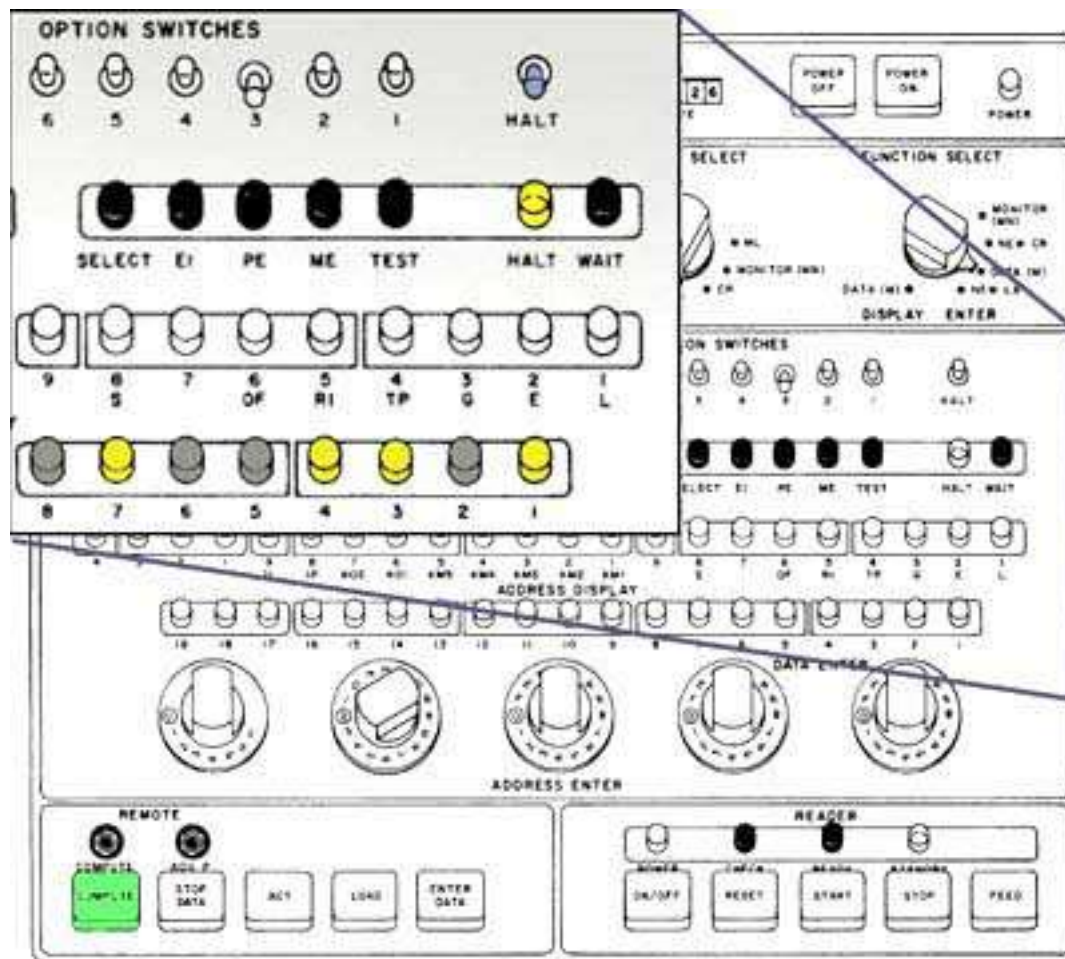
Parecerá increíble, pero esta prueba en mesa era utilísima, no sólo porque detectabas errores lógicos, sino porque te aprendías el programa de tal manera que, una vez que lo probabas de verdad, o sea, en máquina, y no funcionaba como debía, que era lo normal, casi ibas derecho al sitio donde estaba el error. Es una sensación muy curiosa, que yo he tenido muchas veces y que no sé explicar, la de “intuir” en que parte exacta del código está el error... y acertar.

Bueno, al fin ibas a **probar**. La prueba *de verdad*, quiero decir. Generalmente te avisaban para hacer la prueba, ya que muchas veces, sólo viendo cómo se encendían y apagaban las luces de la consola o los dispositivos o cómo se movían las cintas, sabías si había algún problema... igual esto no os lo creéis, pero prometo que es cierto. O, más importante, porque *estando allí, eras capaz de sacar mucho más partido a la prueba* que, como habéis visto, era realmente costosa de hacer.

Me explico con un ejemplo: era bastante corriente que en la primera prueba el programa se metiera en un bucle... infinito, claro. Las cintas se paraban, los discos no se movían y las luces se quedaban encendidas con un patrón determinado. Un bucle, sí, ya (*como siempre*, más bien) pero... ¿dónde? Si estabas allí, podías hacer lo siguiente:

*Uno:* Parar el ordenador. Ya conté que tenía un switch, una palanquita de “Halt” que, cuando la movías, dejaba al ordenador parado en la instrucción donde estuviese.

En la ilustración puede verse un detalle de la palanca Halt y las luces de contenido.



*Dos:* Mirar las luces de la consola que te indicaban la dirección física de la memoria donde está la instrucción que se estaba ejecutando en ese momento. 3E58, por ejemplo.

*Tres:* Ibas a tu programa y mirabas en la parte del listado de la compilación donde tenías la conversión de tu código fuente al código máquina dónde caía la dichosa dirección 3E58, que estaría en una cierta parte del programa, por ejemplo donde estás calculando alguna cosa.

*Cuatro:* Si no quedaba claro qué podía estar pasando (muchas veces, con sólo saber que había un bucle en la zona, ya encontrabas el error), podías ir ejecutando instrucción a instrucción, pulsando sucesivas veces la tecla “Compute”, observando el flujo real que seguía el programa. Podías también averiguar el contenido de ciertos campos importantes para el propio control del bucle.

Si con todo esto no veías qué pasaba, cancelabas la prueba, te retirabas malhumorado a tu mesa y allí te ponías a revisar de arriba abajo el programa hasta que encontrabas el error. Pero era común que encontraras inmediatamente lo que pasaba, por ejemplo, que habías escrito IF A = B y tenía que ser IF A **NOT** = B. Entonces sí que podías continuar con tu prueba para aprovecharla al máximo. Para ello, ibas al código máquina donde estaba esa instrucción errónea concreta y machacabas, a base de usar la consola de forma parecida a la que conté anteriormente, el código de instrucción. Por ejemplo, sustituías un código “EA” -saltar por igual- por un “ED” -saltar por distinto-.

Y ya podías continuar la prueba como si hubieras cambiado el código, que habría que cambiar en el programa final en cualquier caso. Con los debuggers que existen ahora,

esto puede parecer (y es, qué demonios) antediluviano, pero es que **ése era el debugger de la época...** y había que conocerse la máquina de pe a pa para poder utilizarlo... Aquella era una época realmente divertida.

Si tu prueba había fallado, lo que era y sigue siendo lo normal, te llevabas los listados de los ficheros de entrada y de salida a tu mesa, junto con el listado de la compilación, por supuesto, para allí determinar dónde estaba el error y, cuando lo localizabas, volvía a empezar el ciclo perforación-compilación-prueba.

Una anécdota real de aquellos años en la que fui involuntario protagonista os ayudará quizá a comprender las tribulaciones que un programador podía pasar probando sus programas.

Me incorporé yo un lunes al trabajo tras mis merecidas vacaciones veraniegas y me encontré a una colega programadora atribulada que ni siquiera me preguntó qué tal en la playa, si había ligado mucho... o sea, lo normal cuando alguien venía morenito tras sus vacaciones.

Resulta que la pobre mujer llevaba semana y media intentando poner a punto un programa que el banco necesitaba urgentemente... y no había forma de hacer que funcionara. Había probado en máquina media docena de veces, lo había repasado setenta, se lo había contado a todos nuestros compañeros, que habían intentado ayudarla, habían seguido con ella la lógica del programa... y nadie sabía qué le pasaba al muy puñetero. Aparentemente, debería funcionar bien, pero el caso es que se resistía y se iba por *peteneras*.

Lógicamente, me ofrecí para echarle una mano, como el resto de compañeros había hecho ya y ella misma hacía cuando otro se empantanaba, aunque sin muchas esperanzas de encontrar algo que cinco o seis compañeros antes que yo no habían podido ver, pero, relajado como todavía estaba, en lugar de sentarme enfrente de ella en su mesa, me coloqué, de pie, tras ella, mirando desde las alturas el listado del programa que ella me iba contando. Yo era joven y tenía todavía vista suficiente como para leerlo desde la distancia, porque ahora...

De pronto, oteando desde mi atalaya, algo me llamó la atención...

«Oye, María... ¿y ese asterisco?

»¿*Asterisco?* ¿*Qué asterisco?*

»Ése, el que está ahí, en la línea ésa que luego pone un poco más allá ‘GO TO CALCULAR-ALGO.’...

»¿*Asterisco?* ... *Perooooo ¿QUÉ PINTA AHÍ ESE ASTERISCO?*»

En fin. Resulta que en un cierto lugar del programa había un bloque de instrucciones que, muy bien indentado como correspondía a una codificación cuidadosa, pues María era realmente muy cuidadosa, preguntaba si se daba cierta condición y entonces sumaba algo en cierta parte, movía un par de campos, ejecutaba cierto proceso y terminaba con un GO TO a donde fuera que debiera seguir la lógica del programa...

Las instrucciones eran todas correctas y hacían exactamente lo que debían hacer en ese punto, pero había un *ligero problema*: la perforista había perforado un asterisco en la columna 7 de esa tarjeta concreta, o quizás se había colado en el taco de tarjetas en blanco alguna que ya estuviera perforada... El caso es que ese maldito asterisco no debería estar ahí, y su efecto es que convertía a la línea completa en un comentario.

Como a nadie se le ocurriría poner un asterisco en una línea con contenido, nadie veía el dichoso asterisco. Así que tú leías: GO TO tal y tal, pero para el compilador esa

línea no existía, era un mero comentario, así que, como no había tal GOTO, el programa seguía en secuencia fastidiándolo todo, como es natural.

Por pura casualidad vi el asterisco y en quince segundos el problema del asterisco invisible se solucionó, cuando mi compañera, tras cincuenta o sesenta horas de revisar una y otra vez el programa, no lo había visto... ¡ni se podía imaginar que pudiera pasar tal cosa!

Así solían ser las cosas en aquellos pretéritos tiempos, tan movidos e interesantes.

En este punto es muy conveniente, o al menos a mí me lo parece, hacer una reflexión sobre cómo es y cómo se realiza el proceso de prueba de programas... entonces, en el Jurásico, pero ahora también.

En nuestra profesión, la prueba de programas es una actividad importantísima; es donde descubres si tu programa, al que has dedicado horas y horas de trabajo, que has escrito amorosamente en tus hojas de codificación, perforado, revisado y analizado, incluso que te ha hecho despertar por la noche, es el momento, repito, de descubrir si tanto trabajo ha valido para algo o si en realidad lo que tú has programado no sirve para nada. Las pruebas deficientes son las que arruinan el software mejor diseñado: casi todos los problemas actuales con el software vienen de una escasa, incompleta o inexistente planificación de las pruebas, hasta ese punto son importantes.

Para probar, lo que se hace es ejecutar el programa en las condiciones más parecidas posibles a las que se va a encontrar en su vida real en Producción, para luego comprobar si su comportamiento es correcto, es decir, si las salidas que genera se corresponden con lo esperado. Cuando tu programa lee ficheros o, modernamente, Bases de Datos existentes, puedes usarlos como entrada, pero si no existen todavía... entonces debes crearlos de la nada, para que el programa tenga algún dato que llevarse a la boca.

Preparar la prueba es una labor delicada, pues hay que asegurarse de que se prueban todos los casos posibles a los que se pueda enfrentar el programa, y eso no es nada fácil, sobre todo si es el propio programador que escribió el programa el que va a probarlo, que es lo normal.

Una pregunta retórica: En realidad... **¿para qué probamos el programa?** Pues para demostrarnos a nosotros mismos y a nuestros jefes y usuarios que el programa funciona correctamente. ¿Cierto?

Nada de eso.

Las pruebas deben hacerse **no para demostrar que el programa funciona, sino para encontrar errores**. Es decir, una prueba tiene éxito si encuentra al menos un error en el programa. Si no encontramos ninguno, la prueba ha fracasado. Así de crudo. Y de simple. Y de *desconocido*.

Es decir, para probar un programa **lo que tienes que intentar demostrar es que falla**, no que funciona... y eso exige una actitud escéptica ante la prueba, exige intentar *destruir* el programa, demostrar que NO sirve.

Pero... ¿quién prepara y realiza la prueba del programa? El programador que lo escribió. Que generalmente se siente orgulloso de su obra, de su espectacular diseño y las inteligentes soluciones que ha adoptado... Y ahora ¿va a intentar destruir *su obra de arte*? Pocos tienen esa capacidad de mudar su mente. Hacer pruebas efectivas, psicológicamente hablando, es una actitud muy complicada de adquirir.

En general, el programador prepara un juego de ensayo pertinente para hacer pasar al programa por donde él ha previsto, no por donde los datos reales lo llevarán más adelante, para demostrar que esos casos, *específicamente esos*, funcionan correctamente.

Cuando lo ha conseguido, da por buena la prueba y también a su programa. Lo sube a Producción... y luego pasa lo que pasa: que la realidad es caprichosa y no siempre se atiene a lo que él ha diseñado.

Una anécdota al respecto: encargamos un Sistema a una Compañía especializada en desarrollar software. Ellos probaron, lo dieron por bueno, mi gente probó, encontró más errores, se fueron corrigiendo y al final también lo dieron por bueno. Antes de subirlo a Producción se me ocurrió echarlo una ojeada a mí personalmente... no es que no me fiara de las personas de mi departamento, no, es que simplemente quería ver cómo había quedado la cosa...

La primera pantalla, después de pedir ciertos datos, terminaba con un mensaje que era algo así como “Pulse *Enter* para aceptar, o *Escape* para cancelar”. Yo, naturalmente, pulsé una A... y el programa dio un cascotazo de impresión. Todo el mundo se quedó helado y luego me intentaron convencer de que el programa estaba en realidad bien hecho y el error había sido mío, pues no había seguido las instrucciones de la pantalla... Obviamente, devolvimos el Sistema completo a los corrales, que para algo yo era el Jefe, ¿no?, mi gente recibió una charla parecida a ésta que os estoy dando ahora... y comenzaron a aparecer errores y más errores que estaban allá, escondidos, agazapados, pero que nadie había buscado antes.

Algunas semanas después el Sistema quedó *niquelado*, se implantó y rara vez dio problemas hasta su muerte natural por vejez, bastantes años después...

En fin. Retomemos el tema central de este capítulo sobre el sistema de trabajo en los setenta en el punto donde nos habíamos quedado...

...Repitiendo el ciclo perforación-compilación-prueba que cité antes tantas veces como fuera necesario, hasta que sorprendentemente...

**¡El programa funcionaba correctamente!** (o al menos, *eso parecía*, que no es lo mismo). Lo das por válido y das las instrucciones a Producción para que se ejecute cuando le toque. Ahora sí hacías un diagrama precioso de tu programa, usando la plantilla para que quedara bien claro, indicando entradas y salidas, datos a introducir por consola y qué hacer en caso de error: lo normal era poner: “llama al Programador y que resuelva el desaguisado como pueda”, en palabras más técnicas, eso sí. Sea la hora que sea. Y los que sepáis de esto sabéis que, por algún motivo ignoto, las tres de la mañana es una hora muy apetecida por los programas para cascar.

¿He dicho ya que cuando un programa cascaba, se llamaba siempre al Programador, nunca al “Analista Funcional Jefe de Proyecto”?

Al fin, **el programa entra en Producción** y, por lo tanto, pasa a Mantenimiento. El taco de fichas de varios kilos que representa el programa definitivo lo almacenas amorosamente en unas gavetas especiales donde se encuentran los programas en Producción. Algún día habrá que modificarlo para meter algún cambio, así que más vale que esté a buen recaudo.

El listado de la compilación definitiva lo encuadernas con unos artilugios especiales que había para tal fin y lo almacenas, no menos amorosamente, en un armario igualmente especial para guardar listados que entonces se vendía y que supongo que ahora ya no.

En cuanto a la documentación (organigrama, servilleta de bar, apuntes, resultados de la prueba, etc.), la guardas *donde sólo tú sabes*. O mejor, directamente la tiras... Total, lo más probable es que el programa que por fin ha quedado como definitivo no se parezca mucho a los planes iniciales...

Te vas a tu usuario, todo ufano (bueno, casi siempre, el “Analista Funcional Jefe de Proyecto” *va* al usuario, todo ufano...), para decirle que por fin se pueden tratar las Remesas de Letras Compensadas con Vencimiento Fuera de Plaza, que antes había que tratar a mano.

**¡Una Aplicación más en marcha!** ¡Qué buenos somos, hoy dormiremos felices! Sólo hemos tardado uno o dos meses en conseguirlo y, además, en los incontables tiempos muertos, hemos ido escribiendo-compilando-probando los programas para tratar las Remesas No Compensadas con Vencimiento en Plaza, próximamente en sus pantallas... eh, no, en sus pantallas, no. En *sus listados*, más bien.

Es cierto, el proceso se las traía, puramente artesanal, y puede parecer ahora que necesitábamos mucho tiempo para desarrollar el software, pero os garantizo que en la actualidad es imposible poner en marcha ninguna nueva aplicación, por sencilla que parezca, en menos de seis meses... y los que tengáis experiencia, decidme si tengo razón o no.

Además, apenas había programas de utilidad de esos que vienen con el Sistema Operativo y te hacen la vida más fácil. Estaba el Sort y un programa, de cuyo nombre no me acuerdo, que servía para listar el contenido de las cintas... y ya. Resumiendo, cada vez que había que hacer cualquier chapuza en algún fichero, bien porque un programa en Producción había fallado y había que arreglar el desaguizado o por otra causa cualquiera, había que hacer a toda prisa un programa especial *ad hoc* para ello.

Y causas no faltaban, como por ejemplo extraer ciertos registros para hacer lo que fuera con ellos o mandarlos a algún lado, o modificar algún dato en un montón de registros por causas externas, Recuerdo, por ejemplo, que, al poco de aprobarse la Constitución Española en diciembre de 1978, por Decreto-Ley se cambió el nombre de una provincia: la que se llamaba Oviedo pasó a llamarse Asturias. Hubo que hacer muchos programas, sencillos, pero muchos, para, fichero a fichero, donde ponía “Oviedo” en la provincia poner en su lugar “Asturias”.

Y nosotros, los programadores, estábamos un pelín hartos de hacer tanto programa chapuza, no porque nos molestase hacerlos (era nuestro trabajo, así que...) sino porque siempre, siempre, había que hacerlos con prisa. *Mucha* prisa. Si el fichero de Cuentas Corrientes estaba estropeado, os podéis imaginar que no había nada más urgente que hacer en todo el santo Banco que correr a arreglarlo...

Así que decidimos, entre dos o tres compañeros a los que casi siempre nos tocaban estos arreglos de urgencia, escribir un programa que permitiera, de forma dinámica, leer un fichero, aplicarle ciertas condiciones y, si éstas se cumplían, grabar el registro resultante, haciendo antes si fuera preciso algún tipo de operación sencilla: mover a algún sitio un dato literal o desde cierta posición del registro, hacer alguna operación aritmética simple, como sumar o restar algo en un campo numérico, y poco más.

O sea, no era en realidad un programa como todos los demás, sino un **intérprete de comandos** que leía en tarjetas perforadas lo que tenía que hacer... y luego iba y lo hacía.

Nos divertimos muchísimo diseñando y escribiendo esa maravilla de programa, lo pusimos a funcionar, se lo contamos a todos nuestros compañeros, ¡incluso lo documentamos y todo!, y al cabo de poco tiempo la mayoría de problemas de este tenor se los resolvían en Operación ellos solitos, sin molestar a nadie y mucho más rápido, además.

En realidad habíamos inventado, en España y a finales de los 70, un *Lenguaje de Cuarta Generación*... sólo que no lo sabíamos. ¡Cómo lo íbamos a saber, si aún faltaban años para que esos artefactos fueran inventados por los chicos listos de *la cosa*, al otro lado

del Atlántico...!

Creedme: en España y en aquellos años, con nuestros escasos medios, hacíamos cosas increíbles, no sólo en nuestro Banco, sino en cualquier instalación de la época. Lástima que nadie le diera nunca la menor importancia, ni siquiera quienes las hacíamos, y que casi siempre hayan quedado sumidas en el más absoluto de los olvidos...

El caso es que durante todos aquellos años se programaba mayoritariamente en Cobol. Y los ordenadores que mayormente han sido y siguen siendo programados en Cobol hoy en día son los mainframes de IBM. Hoy por hoy, por más que se les ignore olímpicamente, ellos son los que mueven el mundo, y a ellos está dedicado el siguiente capítulo.



## 4 - Los mainframes de IBM

A principios de los ochenta aproveché una buena oferta y salté a otro Banco Nacional. El cambio me supuso una mejora económica (pequeña) y participar en un nuevo y apasionante proyecto. El ordenador de este nuevo Banco, nuevo para mí, quiero decir, era un **IBM 3081** nuevecito, con **dos procesadores**, ¡nada menos!, y **16 Mb de memoria**.

O sea, **un gran mainframe de IBM**, lo más de lo más que había en la época, y casi *lo único* que había como ordenador central. Fijaos bien: pasaba de un ordenador con 64 Kb de memoria a otro con 16.000 Kb... ¡No me podía creer que pudiera existir tanta memoria junta!

Este ordenador era una evolución de la serie IBM 303x, que a su vez lo era del mítico IBM 370, que había sustituido en los primeros años setenta al no menos mítico IBM 360. Su Sistema Operativo era MVS y tenía Bases de Datos IMS (DL/1) y gestor de teleproceso IMS/DC, ambos recién estrenados, que iban a dar servicio al nuevo Sistema Integrado Online de dicho Banco, sistema que había que construir desde cero, claro. Si decía que era un proyecto apasionante no exageraba ni lo más mínimo.

Como se puede suponer, todo el aplicativo se programaba en Cobol, salvo ciertos módulos concretos del Sistema que se escribirían en Assembler, es decir, todo el software de Aplicación de ese Banco (como el de casi todo el mundo) estaba programado en Cobol. Es más, en estos tiempos, que yo sepa, sigue programado en Cobol. Y, en buena parte, siguen funcionando los mismos programas de entonces, remozados, cambiados, modernizados... pero los mismos.

El método de trabajo en estas máquinas ya no era el mismo que el que seguíamos en el otro Banco, con su provento Century 200, según os conté en el capítulo dedicado al equipamiento de los años 70. Entraré más en detalles sobre la forma de trabajar en la época en un par de capítulos.

Los grandes mainframes actuales de IBM, sucesores de aquellos cacharros, programados casi todos en Cobol, siguen siendo las piezas básicas que mantienen al mundo en movimiento. Por ello, en este capítulo voy a describir sucintamente estas máquinas, las grandes desconocidas de la mayoría de la población, e incluso de la mayoría de informáticos de nuevo cuño, que piensan que un servidor Linux de gran potencia es lo máximo... y no por su culpa: es lo que han visto, oído, leído y les han enseñado durante toda su vida. Si estas líneas contribuyen a que conozcan un poco más a estas *bestias pardas* habrán cumplido su función. Por todo ello, no sólo voy a describir los mainframes de los años 80, sino también los de hoy en día.

Veréis que esto no es tan extraño como podría parecer: aunque poseedores de una potencia infinitamente superior, los IBM *z/Series* de hoy en día se parecen mucho, pero mucho, mucho, de cara a sus programadores y usuarios, al venerable IBM/370 de mediados de los años setenta.

Lo primero que hay que mencionar es la **estabilidad** de esta gama de ordenadores. Se han producido, y se siguen produciendo, gigantescos avances en ellos igual que en el resto de ramas de la informática, pero, al estar todos ellos sin excepción dedicados a operaciones críticas, *la estabilidad es la primera máxima de diseño*. Eso quiere decir, por

ejemplo, que un programa escrito y compilado en los años setenta sigue funcionando hoy en día en los ordenadores actuales... *¡sin necesidad siquiera de compilarse de nuevo con cada cambio de versión!*

El código máquina de estos ordenadores es hoy el mismo que hace cuarenta años. Naturalmente, los procesadores son muchísimo más complejos y potentes ahora, hay muchos más códigos máquina de instrucción para hacer cosas cada vez más sofisticadas... pero los códigos originales se mantienen. Es decir, hay una compatibilidad hacia delante que, simplemente, no existe en ninguna otra gama de ordenador, de la marca que sea. Algunas se acercan, es cierto, pero en los mainframes de IBM es la *marca de la casa*. Desgraciadamente, no se puede decir lo mismo de otros sistemas y productos de IBM en otras áreas...

Los mainframes actuales se denominan **IBM z/Series**, al menos así se llaman en el momento de escribir estas líneas, porque ya veis que de tanto en cuando los cambian de nombre, pero no son más que la evolución de los 360, 370, 303x, 308x, 3090, ES/390, etc. Existe un documento, de nombre “*IBM z/Architecture Principles of Operation*”, fácilmente localizable en la página web de IBM, donde tenéis a vuestra disposición TODA la información que deseéis sobre cómo es y cómo funcionan estos Sistemas; se trata del documento básico de estudio obligado para todo Técnico de Sistemas de estos ordenadores y para todo aquel curioso que lo desee.

Sí, todo empezó con aquellos IBM/360 de mediados de los sesenta que revolucionaron la informática para siempre. Lo demás han sido solamente evoluciones y más evoluciones...



Aquí tenemos una imagen de la consola de un ordenador IBM 360 de la NASA en los años 60. Con ellos empezó todo.

Una de las principales diferencias de los mainframes actuales con el ordenador de aquel banco es, además de su potencia varios órdenes de magnitud superior, el tamaño: el 3081 con que operaba aquel banco, más todos sus dispositivos (discos, cintas magnéticas, impresoras, unidades de comunicaciones, etc.) atestaba el solito la Sala de Ordenadores entera, con sus trescientos o cuatrocientos metros cuadrados... y ahora en ese mismo espacio cabrían *decenas* de mainframes.

Son máquinas realmente potentes, según se desprende de los datos que cito a continuación, que son del z10Ec, el más potente de los mainframes de IBM en 2010; ahora seguro que estas cifras han sido superadas:

• **Hasta 64 procesadores** de alta capacidad por ordenador. Son procesadores de cuatro núcleos a 4,4 Ghz de reloj. Esto significa un ciclo de reloj de 0,22 nanosegundos, es decir, 220 picosegundos, seguramente muy cerca de los límites físicos de la electrónica basada en semiconductores. Con una configuración “normal”, de 16 ó 24 procesadores (en España no creo que haya ningún mainframe con 64), es capaz de procesar alguna decena de millares de MIPS (Millones de Instrucciones por Segundo: Millones reales y verificables de Instrucciones cada Segundo, no Millones teóricos, que es lo que se acostumbra a medir en

otro tipo de Sistemas).

- **Hasta 1,5 Tb de memoria** de alta velocidad (eso es *Mucha* memoria... sobre todo comparándola con los pobres 16 Mb de aquel provector 3081).

- **Hasta 1024 Canales de Entrada/Salida.** Son canales de Alta Capacidad y velocidades de transferencia muy elevadas, de muchos Gb por segundo.

Y sin embargo, viendo las características físicas crudas no parece *tan* potente, comparado con máquinas especializadas de procesamiento en paralelo, como la propia “*Blue Gene*” de IBM, uno de los ordenadores más potentes de la actualidad, que tienen miles y miles de procesadores.

Y es que **la diferencia radica en su Software.**

Su **Sistema Operativo** por excelencia es, hablando con propiedad, el único *Sistema Operativo* que existe. Se trata del antiguo MVS, que tras varios cambios de nombre ahora se llama z/OS; IBM ofrecía también por la época, los años 70 y 80, el DOS/VSE, igual que ahora también ofrece Linux para z/Series, ignoro con qué éxito comercial.

Cuando se enumeran las funciones y características que teóricamente debe tener todo Sistema Operativo, sólo MVS (perdón, z/OS; es que todos los que hemos trabajado con él le seguimos llamando “MVS”) cumple todas ellas. Entre ellas:

**1 Disponibilidad total.** IBM dice en su marketing que sus mainframes tienen un 99,999% de disponibilidad, y no sólo es cierto, sino que yo creo que es un 100%. De hecho, los mainframes de IBM sólo suelen apagarse una vez al año, generalmente en Navidades, Viernes Santo, o así, más bien por el prurito de hacer un Arranque en Frío al menos una vez al año (o sea, apagar y encender, vaya) y así limpiar la memoria de posibles zonas muertas, que porque haga realmente falta.

**2 Independencia del Sistema Operativo de las Aplicaciones** (incluso las propias del Sistema). Es decir, absolutamente todos los productos, sean cuales fueran, se pueden instalar en caliente, sin necesidad de apagar y encender nada. Esto reza incluso para el propio Sistema Operativo: casi siempre es posible subir de versión al z/OS sin necesidad siquiera de pararlo. Y desde luego, eso es así para todos los productos, por críticos que estos resulten: IMS, CICS, DB2, RACF, VTAM, etc. Y, por descontado, el MVS (perdón otra vez, el z/OS, la costumbre...) no se cae nunca. Al menos, yo no lo he visto en años.

**3 Absoluta garantía de que una partición no puede acceder a datos de otra,** ni por error, ni a posta. Además, no hay virus, al menos, que yo conozca, para mainframes: la seguridad es máxima.

**4 Capacidad de configurar varias máquinas lógicas (particiones) dentro de una única máquina física.** Con ello se puede trabajar como si tuviéramos tres o cuatro máquinas diferentes, más pequeñas, pero que se comportan a todos los efectos como si fueran máquinas físicas diferentes. Además, la partición no se realiza distribuyendo partes físicas de la máquina (canales o procesadores, por ejemplo) sino potencia: por MIPS. Podemos definir una máquina de 150 MIPS, otra de 400 MIPS y otra de 700 MIPS, por ejemplo. Y, naturalmente, se pueden reconfigurar los tamaños de cada partición en cualquier momento... y sin parar nada, desde luego.

**5 Rendimiento combinado** inalcanzable para cualquier otra máquina. Por ejemplo, un mainframe típico (y no el más grande, repito) es capaz de dar servicio él solo a:

- 500 ó 600 trabajos batch (en la jerga “*mainframera*”, iniciadores), donde se están ejecutando trabajos por lotes: liquidaciones, extractos de cuentas, abonos de dividendo, tareas de backup, pruebas de programas, etc., todo aquello que no necesite hacerse online. No 500 ó 600 *al día*, no. Ni *por hora*. 500 ó 600 *a la vez*, continuamente, todo el día sin parar, todo el año sin parar.

- Varios gestores de teleproceso diferentes (IMS o CICS), sirviendo cada uno algunos millones de transacciones online diarias. Para que os hagáis una idea, un Banco nacional típico puede tener quizá quince o veinte millones de transacciones online diarias; una operadora de Telecomunicaciones, quizá registre 100 ó 150 millones de llamadas diarias, etc. Se procesa online muchísima información hoy en día. ¿Os hacéis una idea de la tasa de transacciones por segundo que supone esta carga?... 200, 300, 400 transacciones por segundo, cada segundo, todos los segundos. Son realmente muchas. Estas transacciones, habitualmente subsegundo, pueden ser servidas por uno sólo de estos gestores, por ejemplo, un buen IMS. Y suele haber otros gestores de Teleproceso dedicados a otras cosas, como la Contabilidad, los Seguros, etc.

- Varios centenares de particiones de time-sharing, el famoso TSO, donde cada usuario ve la máquina como si fuera literalmente para él solo. Por ejemplo, programadores editando y compilando programas, usuarios finales lanzando peticiones de información, operadores gestionando el sistema, etc. Con tiempos de respuesta generalmente subsegundo también.

- ...**SIMULTÁNEAMENTE**. Es decir, **todo lo anterior, a la vez**. Un mainframe de IBM está habitualmente usando su CPU (todas ellas) al 100%. Horas y horas, días y días, meses y meses, sin parar. Es normal verlo incluso al 102%, cosa aparentemente imposible, que es debida al *prefetch* de instrucciones, mecanismo que permite ganar algo de tiempo al solapar la ejecución de instrucciones consecutivas. *Y no pasa nada*. Muchos otros sistemas operativos, al llegar al 80% de uso de CPU comienzan a dar señales de lentitud. Y a partir del 90%, se vuelven inestables. El z/OS, no. Es más, casi todos los que trabajamos con ellos “sabemos” que *cuando están funcionando a menos del 90%, están “vagos”*, como si les costara más de lo normal despachar los diferentes trabajos... Vale, ya lo sé, es una sensación que no tiene base alguna, pero es que la tenemos muchos...

Aunque son sistemas muy fiables por su diseño, también en los mainframes se producen errores y roturas de hardware, como es natural. Con 16 o 24 procesadores que funcionan día y noche es posible que, por muy fiables que sean, **tarde o temprano uno de ellos falle y tenga que ser reemplazado**, y lo mismo ocurre con el resto de componentes: memoria, discos magnéticos, unidades de comunicaciones, etc.

Imaginemos, por ejemplo, una moderna instalación, que puede tener del orden de 800 Tb de espacio en disco disponible, que no sólo se necesita para guardar la información permanente, sino para espacio de trabajo, copias de seguridad, etc. Y 800 Tb son muchos, pero muchos datos: tened en cuenta que aquí no hay almacenadas pelis, ni música, ni las fotos de la familia... tan sólo datos y más datos. Dependiendo de en qué momento se fueron incorporando los diferentes discos a la instalación, estos tendrán una tecnología u otra, girarán más o menos rápido y su capacidad individual estará más o menos entre 250 y 2.000 Gb. Supongamos una media de 500 Gb por disco, lo que puede ser una cifra bastante real.

Pues bien, en esta instalación prototipo, sólo discos de mainframe hay unos 1.600, más o menos antiguos, girando 24 horas diarias a un mínimo de 7.200 vueltas por minuto. Supongamos también que estos discos tienen un MTBF (*Medium Time Between Failures*, o sea, Tiempo Medio Entre Fallos, una medida básica de la fiabilidad de un componente) de 200.000 horas, que no está nada mal. O sea, si tuviéramos un solo disco, podemos esperar, si tenemos suerte, que falle de media cada 200.000 horas, que son... ¡veintitrés años! Pero es que... ¡tenemos 1.600 discos, o más, en nuestra instalación!

Sin hacer muchas cuentas complicadas, que un estadístico sabría hacer (yo lo supe una vez, pero no me acuerdo), podemos darnos cuenta de que en realidad podemos esperar un fallo irreparable de un disco individual cada quizá 80 ó 100 horas, o sea, tendremos indefectiblemente uno o dos discos rotos cada semana, o así. Y puede fallar cualquiera, desde el menos importante, que sólo sirva como espacio de trabajo, al propio disco donde se encuentra el Sistema Operativo.

Pues... **la instalación, sea del sector que sea, no puede pararse por un disco roto**, sea cual sea el que se rompa; y lo mismo si falla cualquier otro componente, por crítico que sea.

En los mainframes es poco menos que imposible encontrar un componente aislado cuya rotura origine una parada total del sistema y toda sustitución de un elemento hardware estropeado se realiza sin necesidad de parar el resto de los componentes (ni los iguales, ni otros distintos) ni, desde luego, el Sistema Operativo. En caso de fallo, dicho Sistema Operativo simplemente marcará ese dispositivo como inutilizable mientras dure la avería, y volverá a utilizarlo cuando ésta se subsane. Aunque no es propiamente un Sistema “*Fault Tolerant*”, se trata de un sistema realmente fiable.

En fin, la gran mayoría de grandes Cajas y Bancos, los diferentes Organismos de la Administración (Hacienda, Trabajo y Seguridad Social, Interior, etc), las grandes compañías industriales y comerciales, sobre todo las que llevan bastantes años en el mercado, casi todas ellas continúan usando mainframes de IBM para sus Operaciones y, que yo sepa, ninguna de ellas tiene la menor intención de cambiar a otro tipo de máquinas.

Porque, además, sorprendentemente, resulta un **Sistema muy barato de administrar**. Con una veintena de Técnicos de Sistemas (muy bien pagados, eso sí) se puede gestionar el parque de mainframes de una gran empresa... cuando con toda seguridad la misma empresa necesitaría centenares de ellos para administrar un Sistema de Información similar basado en, por ejemplo, granjas de servidores UNIX.

Ojo, que no se me malinterprete, **yo no he dicho que un mainframe de IBM sea una máquina barata: en absoluto lo es**. Su coste es muy superior, sobre el papel, al de un Sistema UNIX o Linux equivalente, y no digamos Wintel... Agravado por el hecho de que los programas imprescindibles (Sistema Operativo, Base de Datos, Gestor de Teleproceso, Gestor de Seguridad...) IBM no los vende, sino que son alquilados: no se adquiere una *licencia permanente de uso*, la que permite utilizar de por vida el producto mientras no se actualice... y mientras siga funcionando, claro, sino sólo *el derecho a usar el software durante un cierto tiempo*, generalmente un año, incluyendo el mantenimiento, nuevas versiones, solución de problemas, etc. Y no se trata de productos baratos, precisamente. Pero si nuestra empresa no se puede arriesgar a que una simple actualización de, digamos, el software de la impresora, te deje frita la Aplicación de Cuentas Corrientes o la Aplicación de Pedidos... es la mejor opción.

Además, como dije, no se conocen virus para z/OS. No digo yo que no los haya, sino que de momento no se conocen, así que la seguridad ante intrusiones es máxima. Esto

es debido a que, por su arquitectura interna, es bastante difícil que un hacker pueda encontrar una vulnerabilidad explotable en el Sistema. ¡Y a que ya nadie enseña Ensamblador del z/Series!, exceptuando a la propia IBM, que además cobra una fortuna por ello.

Y recordad, *Aplicaciones escritas en los años 80*, cuando no había prácticamente ninguna alternativa importante a los mainframes de IBM, *siguen funcionando hoy en día sin problemas*, incluso sin siquiera compilar de nuevo los programas, lo que les otorga un Retorno de la Inversión realmente elevado.

...Y no, **yo no trabajo ni he trabajado nunca en IBM**. Considero que no estoy haciendo publicidad alguna, pues todo lo que cuento lo he experimentado yo mismo y visto con mis propios ojitos.

Ahora bien, cuando hablo con muchos compañeros de profesión que nunca han trabajado con estos Sistemas me doy cuenta del enorme nivel de desconocimiento, por no decir desprecio, que existe en la actualidad acerca de esta gama de máquinas y sus aplicaciones. ¡Y eso que, nos guste o no, siguen siendo las que gobiernan el mundo...!

Para acabar el capítulo, no se hubiera podido entender la preponderancia de estos grandes ordenadores y su gran difusión si no fuera por el paralelo desarrollo del lenguaje de programación más utilizado en ellos: el Lenguaje Cobol. En él están desarrollados casi todos los Sistemas de Información que funcionan en estos mainframes.

Pero después de desarrollar tanto y tanto software en Cobol, casi tenemos un disgusto serio al final del milenio...

## 5 - El Lenguaje Cobol. El “Virus del Milenio”

Ahora le toca el turno al otro *cuasi-desconocido* y, en este caso, incluso denostado gran protagonista del mundo de la informática, muy relacionado con los mainframes de IBM: El **Lenguaje COBOL**. Porque en 1980, igual que ahora, el lenguaje por antonomasia en que se programaban los mainframes de IBM es, sobre todo, Cobol.

Bueno, no es en realidad tan desconocido, ya que da trabajo a muchos profesionales para escribir o mantener Aplicaciones escritas en este lenguaje, pero sí lo es, y de qué manera, para los estudiantes de informática de todas las Universidades y Escuelas de todo el mundo. Por lo que yo sé, en prácticamente ninguna se estudia, ni siquiera se suele citar al hablar de los lenguajes de programación existentes.

Y es que el Cobol, más que un lenguaje, es casi *una filosofía de vida*. Ehh, bueno, vale, quizá no tanto, me he dejado llevar por la emoción, o más bien por la nostalgia de tantos años como llevo sin programar en serio...

Cobol (**CO**mmun **B**usiness **O**riented **L**anguage) se definió en 1960 (hace más de cincuenta años), de la mano del todopoderoso DoD (*Department of Defense* de los Estados Unidos). Básicamente, porque estaba harto (¡ya entonces!) de que cada sistema que compraba o subcontractaba viniera escrito en un lenguaje diferente, casi siempre ensambladores que dependían de la máquina, como el caso del NEAT/3 en NCR, lo que originaba grandes gastos para mantener dichos sistemas heterogéneos funcionando.

Así que, ni corto ni perezoso, el DoD participó activamente en el diseño e implementación del nuevo lenguaje, proceso que duró menos de un año y obligó, no inmediatamente, pero sí en el curso de unos pocos años, a que todos los sistemas de gestión que se instalaran en el DoD estuvieran escritos en Cobol y sólo en Cobol.

Este comité, que se constituyó dentro de CODASYL, tuvo como *alma mater* a una de las pioneras de la informática tal y como hoy la conocemos: **Grace Hopper**, a la sazón trabajando en la US Navy de los Estados Unidos y diseñadora de Flow-Matic, lenguaje del que Cobol, digamos, *aprendió* mucho.



Tras toda su vida en la Marina, Grace Hopper, en la imagen anterior al final de su vida laboral, se jubiló como ContraAlmirante de la Armada, siendo seguramente la única mujer en la historia en llegar a ese rango, como contribución a toda una vida dedicada a la informática que tanto la debe, incluyendo el concepto de *bug*, que ella acuñó buscando insectos entre las entrañas del UNIVAC, el primer ordenador, construido a fines de la

década de los cuarenta. Incluso hay una nave de la Marina estadounidense bautizada con su nombre, compartiendo con ella el sobrenombre de “Amazing Grace”, que también a ella se le daba.

Por cierto, veintitantos años más tarde el DoD hizo de nuevo la misma operación de crear de la nada un lenguaje de programación con *Ada*, nombre dado en honor a Lady Ada Lovelace, cuyo retrato de 1838 tenéis a continuación, discípula de Babbage y no sólo la primera mujer “informática”, sino uno de los primeros informáticos.



A diferencia del Cobol, Ada nunca fue un lenguaje muy utilizado, al menos en el mundo comercial; parece que el DoD dejó de exigir sistemas programados en Ada en 1997. Claro que, si le hubieran llamado *Grace*, igual hubiera triunfado... ¡no era nadie, la Señora Hopper!

Todos los fabricantes implementaron más o menos rápido el lenguaje Cobol para sus máquinas, y como consecuencia su conocimiento se extendió rápidamente por los EEUU, pues el Departamento de Defensa era con mucho el principal contratista del país y lo siguió siendo muchos años, y poco después por todo el mundo occidental.

El hecho de que se pudieran programar en Cobol fue utilizado (con éxito) en aquellos años como argumento de marketing en la venta de ordenadores, y poco a poco se impuso como “lingua franca” de los informáticos (*¿Cuál será la lingua franca de los informáticos de hoy en día?, me pregunto. ¿SQL, quizás?*). Otros lenguajes, sobre todo Fortran, se siguieron utilizando en entornos científicos o ingenieriles y aún lo hacen, pero en el entorno de Aplicaciones de Negocio (Contabilidad, Facturas, Proveedores, etc), Cobol se impuso con rapidez.

En este punto me hubiera gustado incorporar una foto de un taco de tarjetas perforadas con un programa Cobol mío, o un listado de una compilación de hace treinta años... pues lo siento: no tengo. En alguna de las muchas mudanzas, acabaron todas en el reciclador de papel. Sin embargo, gracias a Jaume, un lector de ElCedazo que sí que guardaba las tarjetas, más adelante podréis disfrutar tanto de ellas como de los listados resultantes.

Ya desde el principio Cobol define muy claramente la estructura que debe tener un programa mediante una organización en **Divisiones**, de éstas en **Secciones** y éstas, a su vez, en **Párrafos**, en contraposición al otro gran lenguaje de alto nivel de la época, el Fortran, donde se puede definir cualquier cosa en cualquier punto, cosa que a los ingenieros seguro que les venía bien, pero de ninguna manera cuando lo que se escriben son programas de gestión. Hay cuatro divisiones, que son, siempre en este orden:



**IDENTIFICATION DIVISION**, donde se pone la información de identificación del programa: Su nombre, su autor (o autores), fecha de escritura, etc. Esta División sirve más bien a título de comentario.

**ENVIRONMENT DIVISION**, donde se especifica la información de entrada/salida del programa: Qué ficheros usará el programa, cómo son (secuenciales, indexados, etc), cómo se llamarán para el programa y, muy importante, cómo se llaman de verdad en el Sistema, etc.

**DATA DIVISION**, donde se definen... los datos, claro. Los registros de los ficheros (en la FILE SECTION), las áreas de trabajo (en la nunca bien ponderada WORKING-STORAGE SECTION) y, en caso de tratarse de un módulo (que es llamado por otro programa de mayor rango), las áreas de intercambio de información (en la LINKAGE SECTION). La DATA DIVISION es el reino de la **PICTURE**, la cláusula de definición de los datos más potente jamás inventada. Un amigo mío, también de la época de los dinosaurios como yo, comenta siempre que “XML es un standard muy completo, que sólo tiene un fallo, un fallo garrafal: *¡No tener PICTURE!*”.

**PROCEDURE DIVISION**, donde se define lo que es el programa en sí, las instrucciones que conformarán tu programa, con sus lecturas y escrituras, movimientos de campos, cálculos, ejecuciones de rutinas, etc... hasta alcanzar el STOP RUN, donde el programa termina tranquilamente... si antes no ha dado un cascotazo, claro.

En la imagen, un pantallazo de la Procedure Division visto desde un ISPF cualquiera.



Es emblemático en Cobol el uso del **punto** para indicar el fin de la frase, es decir, el fin de sentencia o del párrafo, de la misma forma que el punto se usa en la escritura para terminar una frase o párrafo, salvo que seas Gabriel García Márquez y escribas El Otoño del Patriarca... entonces sólo pondrás un punto cada seis o siete páginas, pero claro, muy pocos somos un genio como el difunto D. Gabriel.

La principal ventaja de usar Cobol reside en su capacidad de **autodocumentación**: escribir un programa en Cobol es prácticamente lo mismo que describir lo que hay que hacer en inglés. Por ejemplo, a continuación transcribo unas pocas sentencias Cobol válidas:

```
READ INPUT-FILE INTO REGISTRO-ENTRADA
  AT END MOVE 'SI' TO INDICADOR-FIN-FICHERO.
```

```
PERFORM PROC-FACTURA
  UNTIL COD-FACTURA NOT EQUAL TO COD-FACT-ANT.
```

```
EXAMINE CAMPO-ALFANUMERICO
  REPLACING ALL 'A' BY SPACES.
```

```
ADD 1 TO CONTADOR-REGISTROS.
```

```

00201 DATA RECORD IS RECORD-IN.
00202 01 RECORD-IN.
00203 05 STUDENT-NAME-IN PIC X(20).
00204 05 STUDENT-CLASS-IN PIC 9.
00205 05 GRADE-1 PIC 9.
00206 05 GRADE-2 PIC 9.
00207 05 GRADE-3 PIC 9.
00208 00 FILE-OUT
00209 LABEL RECORDS ARE OMITTED
00210 DATA RECORD IS RECORD-OUT.
00211 01 RECORD-OUT.
00212 05 STUDENT-NAME-OUT PIC X(20).
00213 05 FILLER PIC X(5) VALUE SPACES.
00214 05 AVERAGE-GRADE-OUT PIC 3-99.
00215 WORKING-STORAGE SECTION.
00216 11 EOF PIC 9 VALUE 0.
00217 11 TOTAL-ORD PIC 99.
00218 PROCEDURE DIVISION.
00219 GPA-REPORT.
00220 *I.O START

```

Por cierto, cuando en España (creo que en Latinoamérica no es así) hablamos “*en Coboliano*”, la pronunciación de los verbos es diferente de la que usamos cuando nosotros mismos hablamos en inglés (bueno, en *esa cosa parecida al inglés* que chapurreamos muchos). Por ejemplo, pronúnciese “RE-AD” en vez de “*Riid*”; “MO-BE TO” en lugar de “*Muuv Tchú*”, “GO-TO” en lugar de “*Gou Tchú*” y, desde luego, “PIK-TU-RE” en lugar de “*Pikchrr*”. Curiosamente, otros verbos sí que los pronunciamos los coboleros como los sajones: “*Compiutt*”, “*Rrait*”, o “*Examinn*”, por ejemplo. Manías de la profesión, supongo... y que me perdonen los lingüistas ingleses por la chapuza fonética: Sorry!

En cuanto a la gestión de ficheros, la del Cobol es muy potente y sencilla de utilizar, y la capacidad de definición y uso de los datos, mediante la famosa cláusula PICTURE, es excelente (siempre “PIC” para los coboleros, pocos escribían PICTURE en un programa de verdad: es más largo, y como había que perforarlo todo...).

Una curiosidad Cobolera: La siguiente sentencia es una sentencia válida en Cobol:

### **NOM-PARRAFO. GO TO.**

Go To... *¿adónde?*, preguntaréis. Porque una sentencia de bifurcación (un GO TO) que no dice adónde se produce la susodicha bifurcación... parece un poco raro, ¿no?

Y es que una característica del Cobol, que no se da en casi ningún otro lenguaje de alto nivel, es la capacidad de, utilizando de forma legal las sentencias del lenguaje, *modificar el propio código máquina de una instrucción del programa*. Para ello está disponible la (odiada) sentencia “ALTER”. Por ejemplo, podríamos codificar:

### **ALTER NOM-PARRAFO TO CALCULAR.**

Esta sentencia cambiaría el código del GO TO en NOM-PARRAFO para apuntar a la dirección de “CALCULAR”. Cuando, por su orden de ejecución normal, el programa alcance la dichosa sentencia GO TO, el programa ahora bifurcará a “CALCULAR”.

Naturalmente, puede haber varias sentencias ALTER que vayan modificando la bifurcación para que apunte a diferentes puntos del programa en función de las condiciones que se vayan dando... lo que puede ser una forma muy eficiente de programar... y un auténtico quebradero de cabeza para luego seguir lo que hace el programa.

Como podéis imaginar, el uso de esta sentencia está tácitamente prohibido por el manual implícito de buenas prácticas del *Cobolero medio* (y por las normas de programación de las empresas que las tienen, que son muchas, por no decir todas), lo que no impide que algún inteligente lo use de tanto en cuando... para fastidiar, claro.

Sin embargo, en el momento de la definición del lenguaje en 1960, era una sentencia lógica, potente y bastante usada en los primeros años, en sustitución de la sentencia PERFORM (la de ejecutar una rutina y, al acabar, proseguir el proceso por la instrucción siguiente a la propia PERFORM, como el DO de Fortran y otros lenguajes). Ahora nos parece raro e incomprensible, pero los ordenadores de entonces tenían tan poca la potencia que todo valía para ahorrar algunas instrucciones...

En la ilustración se pueden observar las dos diferentes maneras de programar esta llamada, donde hay codificadas dos formas distintas, pero completamente equivalentes, de ejecutar la subrutina -A- en Cobol.

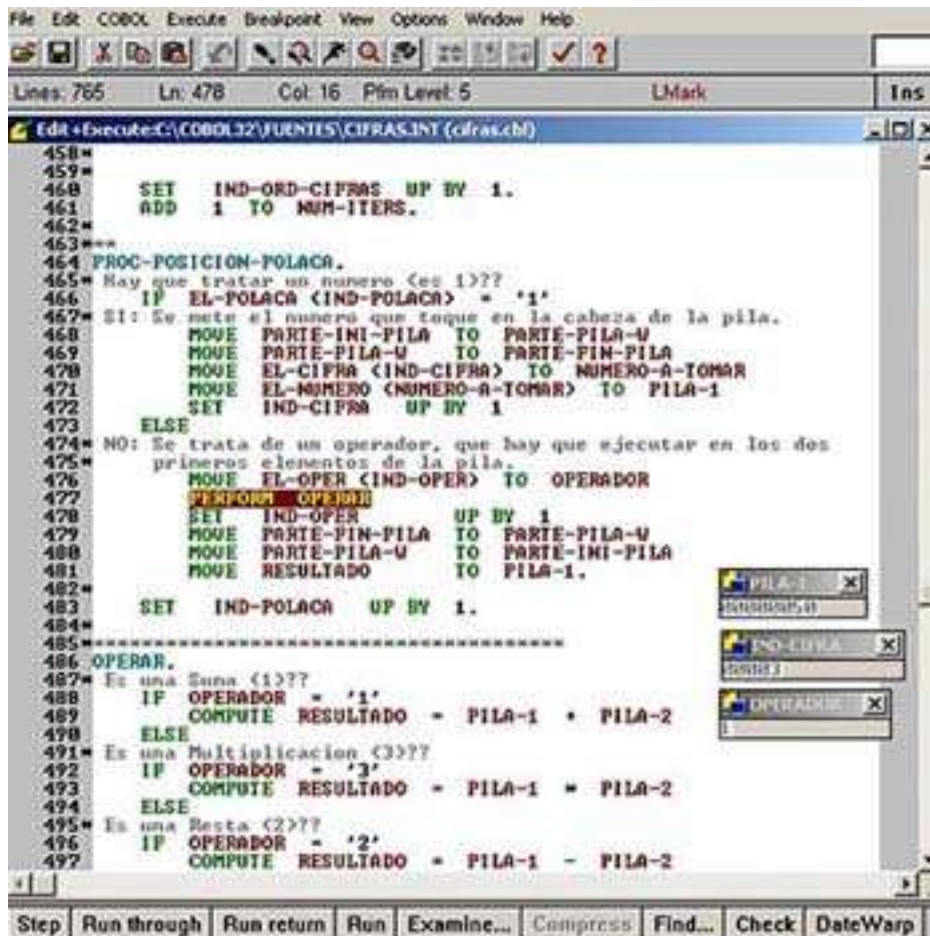
PERFORM A THRU A-END. . . . A. instrucciones a realizar A-END. EXIT.	ALTER A-END TO SEGUIR. GO TO A. SEGUIR. . . A. instrucciones a realizar A-END. GO TO.
--	--

Cuando los ordenadores tenían muuuy poca capacidad, codificar de la forma de la derecha ahorra espacio, instrucciones y ciclos de máquina, por lo que era bastante usada, incluso recomendada por algunos manuales de buenas prácticas de la época. Había que adaptarse a lo poco que había.

Cobol se fue adaptando, con el tiempo, a la tecnología de cada momento, incorporando fácil acceso a las Bases de Datos que fueron apareciendo (IMS/DL1, Total, Adabas, IDMS, DB2, etc.), Gestores de Teleproceso (IMS/DC, CICS, etc.) y a otros ambientes y sistemas operativos (UNIX, Linux, etc.), aunque no tanto al mundo del PC con Windows.

Existen diversos Compiladores de Cobol para PC desde mediados de los años 80, así como un entorno completo de Simulación para programar, compilar y probar completamente en un PC programas Cobol escritos para el entorno mainframe.

Este entorno permite simular no sólo el propio compilador, sino todos los productos del mainframe, como CICS, IMS o DB2, con un potente debugger, ¡incluso permite simular la ejecución de rutinas escritas en Assembler! permitiendo una productividad muy alta en el desarrollo y las pruebas y una total (bueno, *casi total*) compatibilidad. A continuación, una imagen del debugger de Cobol Microfocus.



```
File Edit COBOL Execute Breakpoint View Options Window Help
Lines: 765 Ln: 478 Col: 16 Pgm Level: 5 LMark Ins
Edt+Execute:C:\COBOL32\FUENTES\CIFRAS.INT (cifras.cbl)
458*
459*
460*   SET   IND-ORD-CIFRAS  UP BY 1.
461*   ADD   1 TO NUM-ITERS.
462*
463***
464* PROC-POSICION-POLACA.
465* Hay que tratar un numero <es 1>??
466* IF EL-POLACA <IND-POLACA> = '1'
467* Si: Se trata el numero que toque en la cabeza de la pila.
468*   MOVE PARTE-INI-PILA TO PARTE-PILA-U
469*   MOVE PARTE-PILA-U TO PARTE-FIN-PILA
470*   MOVE EL-CIFRA <IND-CIFRA> TO NUMERO-A-TOMAR
471*   MOVE EL-NUMERO <NUMERO-A-TOMAR> TO PILA-1
472*   SET   IND-CIFRA  UP BY 1
473* ELSE
474* NO: Se trata de un operador, que hay que ejecutar en los dos
475* primeros elementos de la pila.
476*   MOVE EL-OPER <IND-OPER> TO OPERADOR
477*   PERFORM OPERAR
478*   SET   IND-OPER  UP BY 1
479*   MOVE PARTE-FIN-PILA TO PARTE-PILA-U
480*   MOVE PARTE-PILA-U TO PARTE-INI-PILA
481*   MOVE RESULTADO TO PILA-1.
482*
483*   SET   IND-POLACA  UP BY 1.
484*
485*****
486* OPERAR.
487* Es una Suma <1>??
488* IF OPERADOR = '1'
489*   COMPUTE RESULTADO = PILA-1 + PILA-2
490* ELSE
491* Es una Multiplicacion <3>??
492* IF OPERADOR = '3'
493*   COMPUTE RESULTADO = PILA-1 * PILA-2
494* ELSE
495* Es una Resta <2>??
496* IF OPERADOR = '2'
497*   COMPUTE RESULTADO = PILA-1 - PILA-2
```

También existen productos que implementan Cobol en PC que permiten desarrollar aplicaciones completas, y no sólo aplicaciones puramente batch.

Se han definido e implementado extensiones para soportar Windows (u OS/2, en su día), el paradigma Cliente/Servidor, el de Orientación a Objetos... pero ciertamente *nadie en sus cabales escribe una aplicación Cliente-Servidor, ni menos aún Web, en Cobol.*

Algún iluminado, inasequible al desaliento, llegó a proponer que al Cobol Orientado a Objetos (OO Cobol) se le cambiase el nombre a **“ADD 1 TO COBOL GIVING COBOL”**, que es la traducción literal a *Coboliano* del término “C++”, puesto que éste último nombre de lenguaje había tenido tanto éxito en este entorno del OO... Pero parece que pocos entendieron la gracia y, al final, no tuvo éxito tal propuesta. ¡Qué cosas! Se quedó con “OO Cobol”, pero de todos modos no tuvo mucho éxito, que digamos. Bueno, ni mucho ni poco: nada en absoluto.

Cierto, el Cobol es un lenguaje realmente “*verbose*”, o sea, lenguaraz: seguramente para hacer las mismas tonterías de las sentencias anteriores en C (con o sin los signos de la suma detrás) o en Java se requerirían quizá tres líneas cortitas en total... pero esa “*verbosidad*” redundaba en un buen entendimiento de lo que hace un programa Cobol cuando lo mira un programador distinto de aquel que lo escribió (o mejor, un becario despistado al que hemos dado un curso de tres días de Cobol)... *veinte años después*. Y eso, en una instalación informática de verdad, con decenas de miles de programas y varios millones de

líneas de código en Producción, cotiza muy, pero que muy caro.

Ya he comentado que muchas Aplicaciones llevan veinte años o más en Producción y es lógico pensar que han debido sufrir cambios durante este tiempo: naturalmente que sí, y muchos. Cambios legales, nuevas necesidades de la compañía, nuevos productos, etc., obligan a cambiar funcionalidades, ficheros, bases de datos... y todo cambio termina obligando a cambiar a su vez uno o varios programas. O muchos. ¡O todos!

En estas condiciones, poder entender rápidamente la funcionalidad de un programa, localizar el lugar donde debe ser modificado, modificarlo y que además funcione correctamente según las nuevas especificaciones, y en un tiempo razonable, es el factor clave en la elección de un lenguaje de programación. En esto Cobol se lleva la palma.

Me da a mí la sensación, tras hablar con muchos colegas, de que el paradigma sobre el mantenimiento de las aplicaciones escritas en C, por ejemplo, es que *resulta más barato reescribir el código de nuevo*, cuando hay que hacer una modificación de cierta entidad, que *tocar el código existente...* porque casi siempre el programa resulta prácticamente incomprensible para un programador distinto del que lo escribió (y en ocasiones, incluso para *el mismo que lo escribió*). Esto no se puede tolerar en una gran instalación.

Que me perdonen los fans del C, o del Java, o del Perl, o del propio Ada, pero esta es mi opinión... y la de muchos otros colegas, también.

Para terminar el capítulo dedicado al Cobol y gracias a la colaboración de nuestro amigo Jaume, que en un comentario en una de las entradas de la serie en Elcedazo nos decía que él guardaba como oro en paño las tarjetas perforadas de un programa Cobol del año de la polka y también algunos listados, podemos disfrutar aquí de ese incunable, pues es difícilísimo encontrar imágenes de estos ancestrales artilugios en la red o en cualquier otro sitio.

Jaume tuvo la enorme amabilidad de, en primer lugar, buscar las tarjetas del programa entre el uniforme del Colegio y los cromos de *Vida y Color*, supongo, y, después, fotografiar o escanear las tarjetas y los listados, que me envió para su inclusión en una nostálgica entrada en la serie y poder, al fin, compartirlos aquí con todos vosotros. A continuación podéis admirar esas maravillosas fotos.

**¡Muchas gracias, Jaume, en nombre de todos los nostálgicos de aquella época!** Y no sólo de los abuelos, sino, me consta, también de otros muchos interesados en conocer algo más de la Historia de la Informática española.

*Las sempiternas tres primeras tarjetas de un programa Cobol: la Identification Division*





Como podéis ver, las seis primeras posiciones son la *página/línea*, que podía usarse para reclasificar el programa en caso de caída y consecuente *descolocamiento* del taco de tarjetas; usualmente se numeraba cada página de la Hoja de Codificación donde se escribía el programa con un número de tres cifras: 001, 002..., y luego cada línea de la Hoja de Codificación, con números consecutivos, pero de diez en diez: 010, 020... para dejar espacio por si más adelante hubiera que insertar alguna otra tarjeta entre medias, cosa de lo más normal.

La columna 7 sólo se usa para comentarios, cuando lleva un asterisco, o para indicar la continuación de la línea anterior, si lleva un guión.

A continuación viene el texto codificado en Cobol, desde la columna 8 a la 72, ambas inclusive. La columna 8 es del así llamado “*margen A*”, donde se escriben los nombres de párrafo, sección o división; las cláusulas e instrucciones se escriben a partir de la columna 12, el “*margen B*”, y efectivamente, muy imaginativo no era el nombre, no.

Por fin, en las ocho últimas posiciones, se perfora el nombre del programa, que en este caso es DFASEM17.

El nombre del programa en realidad se toma siempre de la sentencia **PROGRAM-ID** de la IDENTIFICATION DIVISION. Lo que ponga o deje de poner en las columnas 73 a 80 es tomado por el compilador tan sólo como un mero comentario.

Sin embargo, la costumbre de casi todo el mundo era perforar allí el nombre del programa, y no, la perforista no tenía que hacerlo en cada tarjeta, sino que programaba la perforadora para que automáticamente lo hiciera ella sola en cada tarjeta del programa; con ello se podía identificar rápidamente un taco de tarjetas... o también saber a qué programa pertenecía una tarjeta suelta que te encontrabas revoloteando por los pasillos, cosa que, por increíble que parezca, sucedía de tanto en cuando.

A continuación, las tres siguientes fichas del programa. La primera (“JAIME”) es en realidad el *AUTHOR* del programa; la ficha AUTHOR es la última de la foto anterior.

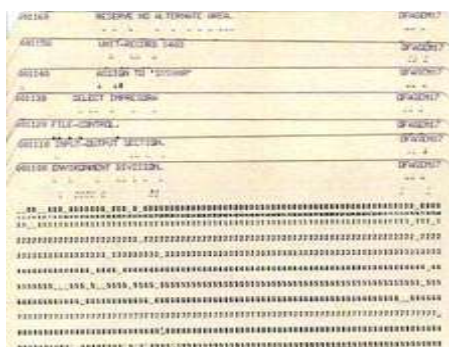


Observad la fecha de escritura del programa, en la cláusula “DATE-WRITTEN”: Febrero de 1975. Yo entonces estaba en Tercero de Carrera y no había comenzado aún a trabajar, hasta septiembre de ese año no comencé mi vida laboral: Jaume es, pues, más *experto* todavía que yo...

Podéis ver también las perforaciones que corresponden a cada carácter. En este caso ha habido suerte: la tinta de la máquina de perforar estaba bien y se han impreso, en la parte superior de la tarjeta, los caracteres que están físicamente perforados en ella. En muchas ocasiones la tinta estaba agotada y no se imprimía nada, por lo que, si querías saber lo que estaba allí perforado, no te quedaba otra opción que mirar los agujeritos al trasluz, para traducir mentalmente las perforaciones a caracteres: todos éramos muy duchos en estos menesteres.

Por ejemplo, la “A” son dos perforaciones, una en la fila 12 (la primera de todas, que no lleva caracteres impresos) y otra en fila 1; la “B”, dos perforaciones en filas 12 y 2; la “T” son dos perforaciones en fila 0 y fila 3; los números son sólo una perforación en la fila correspondiente (el cero en la fila 0, el cinco en la 5, etc), los caracteres especiales solían tener tres perforaciones... Cada carácter de los 256 posibles tenía su código, incluso los no imprimibles: se trataba del Código Hollerith.

A continuación, parte de la Environment Division, donde se especifican los ficheros que usa el programa y qué tipo de fichero es cada uno: una cinta magnética, un fichero en disco, la impresora, etc.





Fijaos en la ficha “ASSIGN TO 'SYS008'”: para el Sistema en que trabajaba Jaime ese nombre, *SYS008*, identificaría a la impresora del Sistema, y no es que yo lo sepa, lo deduzco porque el nombre del fichero es “IMPRESORA”: a los programadores de entonces nos gustaban las cosas claras. Esa impresora, además, seguro, seguro que hacía impresión directa, sin pasar por Spool, es decir, cuando el programa ejecutaba un “WRITE LINEA”, esa línea en concreto se imprimía en ese mismo momento en el papel pijama. Si la impresora no estaba lista, el programa se quedaría parado, esperando a que lo estuviera hasta poder realizar la impresión.

También se dice en la Environment Division cómo se llaman los ficheros de verdad y otras definiciones generales; por ejemplo aquí es donde los europeos le avisamos al compilador de que “DECIMAL-POINT IS COMMA”, para que cambie el uso yanqui de puntos y comas en la representación de los números, que es la opción por defecto, y lo entienda como lo entendemos los europeos.

Una de las cosas que se decían en la Environment Division es si algunos ficheros comparten o no el área de lectura o escritura (mediante la cláusula RESERVE...). Eso, que ahora seguro que suena a gaélico del Siglo XI, es posible hacerlo en Cobol: como la memoria era tan limitada, era relativamente normal decirle al compilador que utilizara el mismo espacio físico para guardar los buffers de entrada o salida de dos o más ficheros distintos, ahorrando así alguna que otra Ka, que te podía venir de miedo para conseguir que tu programa cupiera entero en la memoria.

Naturalmente, era tu responsabilidad que sólo uno de los ficheros que compartían área de Entrada/Salida estuviera abierto en cada momento o, si no, los resultados serían *impredecibles*, que es el término que utilizaban los manuales de la época... o sea, que tenías garantizado un cascotazo de impresión y, además, de esos que no había forma humana de saber qué demonios le había pasado al programa para que hubiera cascado. Y esto, creedme, lo digo por experiencia...



En el bloque de tarjetas anterior tenéis parte de la Data Division, donde se definen los datos: en la File Section se definen los ficheros y sus registros y en la Working-Storage Section se definen las variables y campos de trabajo que usará el programa.

Estamos en el *reino de la PICTURE*. Jaime prefería usar PICTURE en lugar de PIC, como casi todo hijo de vecino hacía; debía ser de los pocos tipos raros que hacían

esto, aunque lo más seguro sea que el compilador que tenía ese sistema concreto no admitiera la abreviatura *PIC*, que de todo había: en los sistemas pequeños, con muy escasa capacidad de almacenamiento y no digamos de memoria, era relativamente normal que los compiladores sólo usaran un subconjunto de instrucciones, el mínimo imprescindible para poder escribir programas funcionales, pues así ahorran espacio y tiempo de compilación.

Por ejemplo, yo me las tuve que ver en cierta ocasión con un compilador que no admitía, entre otras, la cláusula “PERFORM” (o sea, la instrucción equivalente al *DO* de Fortran, o a una *function* de C). En ese sistema, la única forma de ejecutar una rutina era a base de *GOTO*’s, con o sin programación estructurada... ¡Como para prohibirlos!

Y, por fin, parte de la PROCEDURE DIVISION:



Es aquí, en la Procedure Division donde se define el proceso que debe hacer el programa, es decir, lo que mayormente es el programa en sí. El reino de la “*MOVE*”, vaya. Los programadores viejos nos referíamos coloquialmente a *programar* como “*Darle a la MOVE*”, por algo sería.

Observaréis el uso del *GO TO* sin el menor pudor; en aquellos años nadie programaba en España, ni casi en el mundo, usando el paradigma de la Programación Estructurada, así que era así como todo el mundo programábamos: no sabíamos otra manera... y tan felices. Felices hasta que te tocaba modificar un programa de los que había hecho “*El Rey del GOTO*” (en cada instalación había al menos uno), y entonces te acordabas del inventor del *GOTO*... y no para felicitarle, precisamente.

Por fin, como seguramente sabéis, en los años setenta y ochenta del siglo pasado no se había inventado la internet, así que, cuando nos aburríamos, lo que a veces pasaba, aunque poco, en vez de mirar estupendos blogs de ciencia, como ahora hacemos, nos dedicábamos a hacer monerías...

Y a continuación podéis disfrutar de las dos obras de arte... el taco de tarjetas del programa y el listado resultante.



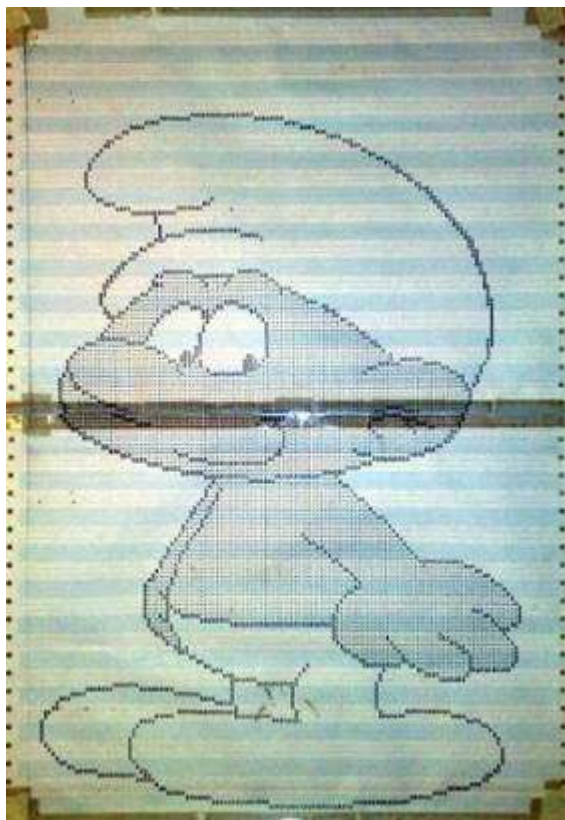
Sobre el listado podéis observar el taco de fichas del programa: quizás tenga quinientas o seiscientas tarjetas y debe pesar alrededor del kilo y medio.

Notaréis sin duda la marca que ha dejado la gomilla con la que todo taco de programa que se precie estaba sujeto para que no desparramaran las tarjetas, gomilla que, con toda seguridad, ha resultado destruida por el implacable paso del tiempo.

Jaume, ya lo veis, dibujaba pitufos... nosotros éramos más de pintar escudos del *Madri* y del *Aleti*... bueno, y había uno maño que los hacía del Real Zaragoza. Pero es que nosotros estábamos en Madrid; me imagino que en cada ciudad estaría más trabajado el escudo del equipo local de fútbol (no existía otro deporte que pudiera hacer sombra al fútbol entonces... bueno, ni ahora tampoco) y otros motivos regionales alegóricos.

Sorprende lo bien conservado que se mantiene el listado: treinta y tantos años después, la tinta sigue perfectamente nítida, pero es el que el papel pijama de entonces era de excelente calidad, de alto gramaje, pues en otro caso no hubiera podido aguantar los brutales tirones que daban las impresoras de la época, que no eran precisamente muy suaves cuando funcionaban a todo trapo.

Y ya, por fin, podéis disfrutar completo del resultado final de tantos desvelos:



Como se ve claramente, lo que peor ha aguantado el paso del tiempo, aparte de la difunta gomilla es... ¡el papel celo!

Espero que esta vaharada de *cobolera* nostalgia os haya gustado tanto como le ha gustado a un servidor. Y, otra vez: **¡gracias, Jaume!** Los arqueólogos de la informática te estamos eternamente agradecidos.

## El Virus del Milenio

Para terminar este capítulo, y dado que tiene indudablemente mucha relación con él, quizá os acordéis del publicitadísimo “Virus del milenio”, cuando hubo que hacer grandes inversiones en informática derivadas de la inevitable llegada del año 2000, exactamente al día siguiente del 31 de diciembre de 1999. Por cierto, esas inversiones no fueron nada comparadas con lo que hubo que hacer en Europa el 2001 con la llegada del Euro... ésa sí que fue de aúpa, pero ésa es otra historia y será contada en otro momento.

Primero los consultores nos amenazaron con el fin del mundo, luego nos echaron la culpa a los inútiles informáticos por nuestra descomunal falta de previsión, se vendieron libros y revistas, se programaron seminarios, conferencias y charlas por todas partes, se escribió mucho y se habló más... y luego, cuando el 1 de enero del 2000 todo funcionó, nos tildaron de alarmistas, derrochadores y de que no era para tanto...

Y, total, todo porque no se cayó ningún avión, ni explotó ninguna central nuclear, ni siquiera hubo un triste Banco que dejara de procesar sus tristes transacciones. En la página de la Wikipedia inglesa sobre el tema (“*Year 2000 Problem*”) podéis comprobar, si lo deseáis, la lista de “grandes” errores que acontecieron en el mundo el 1 de enero del 2000.

Cuatro tonterías... ¡Imaginaos qué nos hubieran llamado si alguna catástrofe de verdad hubiera ocurrido!

Las consecuencias, efectivamente, fueron mínimas y todas ellas fueron rápidamente solventadas, así que mucha gente pensó que los informáticos habíamos engañado a todo el mundo con el fin de conseguir mucho dinero para corregir un supuesto error que, en definitiva, no era para tanto...

Pues... **Sí** que fue para tanto.

Yo os aseguro (lo viví en primera persona, como muchos otros), que fue un trabajo exhaustivo y agotador, revisando, cambiando, probando y poniendo en Producción muchos miles de programas interrelacionados entre sí... una labor de chinos, y donde el control de cambios era vital para asegurar que una aplicación cambiada siguiera “hablando” con las que aún no lo habían sido y viceversa. Fue tremendo.

**Y todo este trabajo, para que todo siguiera igual.**

Bueno, vale, también se aprovechó para cambiar algunas cosas, incluso para tirar las realmente viejas y hacerlas nuevas. Pero en muchísimos casos, no: nos limitamos a cambiar el código para tratar la fecha sin introducir ningún cambio adicional. En definitiva, dinero “perdido”, según los administradores de las empresas. Y con razón, pensaréis... *¿Con razón?* No lo creo. Veamos:

Todo el asunto fue motivado por la “manía” de muchos informáticos (*¿muchos...?* no: **¡todos!**) de almacenar las fechas en los Sistemas de Información **con el año en dos dígitos**. Así, por ejemplo, el 7 de febrero de 1978 se almacenaba como “780207”, en vez de “19780207”. Por cierto, no como “070278”, pues de esta forma las fechas no quedan ordenadas según el natural orden cronológico; a la hora de imprimirlas, se dan la vuelta, pero internamente están siempre Año-Mes-Día. Lógicamente, el día más alto posible era el “991231”. El día siguiente, el uno de enero del 2000, se vería como “000101”... que es anterior a todas las demás fechas hasta entonces, pues es la más pequeña posible.

En sí, si nos fijamos bien, no es que pase nada especial... salvo por los programas que necesitan ordenar movimientos por fechas, o calcular diferencias entre fechas (como las liquidaciones de cuenta, o los cálculos de la fecha de caducidad, por ejemplo). Es decir, muchos. Pero *muchos*. En realidad, todas las aplicaciones contables (las que manejan datos de dineros, o sea, casi todas en las empresas) están afectadas en mayor o menor medida. No quedaba, pues, más remedio que modificar todos los ficheros y bases de datos para aumentar el tamaño de las fechas, después cargar los nuevos ficheros con los datos antiguos (añadiendo un “19” en las dos primeras posiciones del año) y listo. Fácil, *¿no?*

**Pues no.** El problema es que esto afectaba a muchos millones de ficheros y bases de datos en todo el mundo... y muchos cientos de millones de programas que los leían y escribían.

En este punto uno no puede dejar de preguntarse: *¿Cómo es que los informáticos del mundo mundial, tan listos ellos, no vieron venir este hecho, y se empeñaron, años y más años, en almacenar el año en dos dígitos?* Es más, debió tratarse efectivamente de un virus de alcance universal, porque no se escapó nadie, ni en España, ni en Latinoamérica, ni en EEUU, ni en Europa, ni en la India, ni en ninguna parte del orbe.

Pues hay varias razones para tan *criminal* proceder:

En los años 70 y primeros 80 a nadie en su sano juicio se le podía ocurrir que las



Aplicaciones que diseñaba con los años en dos dígitos pudieran resistir 20 años o más sin que una nueva y más moderna aplicación sustituyera a la que estabas escribiendo. Por ejemplo, en los primeros 80 en el Banco ya estábamos desarrollando completo un nuevo Sistema Bancario que sustituiría a otro con menos de diez años de vida, y lo mismo ocurría en muchas instalaciones. La Informática era joven y en unos pocos años habíamos ya sustituido muchas Aplicaciones, así que, ¿cómo pensar que la cosa que poníamos en marcha en 1981 pudiera durar hasta 2014... y *contando*?

Y es que, por estúpido que parezca hoy, **almacenar las fechas con ocho dígitos en lugar de seis era caro. Carísimo.** ¿No os lo creéis? Hagamos unas pequeñas cuentas. Un Banco español de entonces podía tener quizás un millón de clientes, y cada cliente tenía una o varias cuentas que tienen algunos apuntes cada mes, quizá cinco o seis.

En cada uno de estos apuntes (operaciones), hay al menos tres fechas: la de operación, la de contabilización y la de valor. Y muchas más en el resto de la información de la cuenta: la fecha de apertura, la de liquidación, la de cancelación, la de entrada en mora, etc.

Utilizar ocho dígitos para almacenar las fechas en lugar de seis ocuparía 6 caracteres más por cada apunte. Una fruslería, ¿no? Bueno, sigamos con nuestras cuentas... Con varios centenares de millones de apuntes almacenados, la diferencia supondría apenas cinco o seis Gigabytes de almacenamiento en disco. Y no sólo había fechas en los apuntes. **¡Había fechas por todas partes!** Montañas de fechas de todo pelaje, esparcidas por toda clase de ficheros y Bases de Datos.

Por cierto: ni tan siquiera la “*Fecha de finalización de un préstamo hipotecario*” estaba guardada con el año en cuatro caracteres en los años ochenta... **¡El plazo máximo al que se concedía un préstamo eran doce años!** Y de los tipos de interés, ya ni hablamos, todos eran de *dos cifras*. Para que os quejéis los jóvenes... En fin, el caso es que había toneladas de fechas que almacenar.

¡Pero es que los discos de la época sólo tenían algún ciento de Megabytes cada uno, y eso que ya no eran removibles, sino fijos! O sea, que por el mero *capricho* de guardar espacio en las fechas para tener los años de cuatro cifras, sería necesario adquirir alguna decena más de discos... que eran bastante caros, como podéis imaginar...



Por ejemplo, el IBM 3380 DASD de la imagen, la unidad de disco más avanzada y capaz de la época, que en lo que a almacenamiento se refiere marcó un hito, era capaz de almacenar nada menos que ¡10 Gb!, en *solamente cuatro* armarios realmente aparatosos, como podéis comprobar, con varios discos cada uno, que ocupaban un espacio enorme, quizá seis o siete metros de largo por dos de alto y uno de ancho, más el área de servicio de

al menos otros 60 cm. por cada lado. Y todo ese espacio para diez míseros (y carísimos) Gb de datos.

En definitiva: el dispendio de valioso espacio en disco necesario para almacenar los años con cuatro dígitos **no servía para nada**: ni a corto, ni a medio, ni a largo plazo: hasta quince o veinte años más tarde no se notarían las supuestas ventajas de tener las fechas así almacenadas... si la Aplicación seguía aún funcionando para entonces, claro, cosa por entonces inimaginable.

No había forma, en los años setenta y ochenta del pasado siglo, de justificar económicamente usar cuatro dígitos para el año. Las cuentas no salían, así de sencillo.

Mmmm. Vale, de acuerdo, lo acepto... **También nosotros, los de la profesión, tuvimos parte de la culpa**: ya a principios de los noventa empezó a ser evidente que las aplicaciones que se desarrollaban entonces deberían durar más de diez años, el coste del almacenamiento había ido bajando muchísimo... y sin embargo muchas se siguieron escribiendo con los años en dos cifras. Sobre todo por inercia (*“como toda la vida se han almacenado en dos, ni te paras a pensar si no sería mejor hacerlo de otro modo”*) y un poco por compatibilidad (*“si hay que comunicarse con otra aplicación que ya funciona, hay que adaptarse a su formato”*).

Sólo a partir de 1995 o 1996 se desterró definitivamente la costumbre de almacenar las fechas en seis caracteres... tarde, muy tarde.

Pero... se ve que aprendemos rápido de nuestros errores: **ya hay gente muy sesuda preocupándose del problema del año 10.000**, *¡exigiendo que los años se almacenen en cinco cifras!*, para evitar que los tataranietos de los tataranietos de... nuestros tataratataranietos tengan que lidiar con el mismo problema dentro de 8.000 años.

Está claro que los humanos no estamos nunca contentos con nada... Si no os lo creéis, buscad en la Wikipedia inglesa la página sobre *“Year 10,000 Problem”* y veréis si tengo razón o no la tengo... Y añado yo que, puestos a ser previsores, podríamos almacenar los años con *seis* cifras, y así también solucionaríamos, de paso, el problema del año 100.000... En fin.

En definitiva, en los años setenta y ochenta el lenguaje Cobol lo usábamos en casi todas partes para escribir nuevas y más capaces aplicaciones que permitieron a las empresas, entre ellas a los Bancos como aquel en que yo trabajaba en la época, cambiar radicalmente su funcionamiento.

E, indefectiblemente, también el método de trabajo en sí que usábamos nosotros, los informáticos de postín en estos fantásticos Sistemas durante los años ochenta, cambiaba deprisa...

## 6 - El método de trabajo en la década de los ochenta

Efectivamente, a principios de los ochenta me cambié a otro Banco que estaba comenzando a escribir de nuevo su Sistema Informático. Este nuevo Banco tenía un Sistema funcionando, similar al que os describí en el capítulo dedicado al equipamiento de los setenta, aunque de otra marca, y había decidido cambiarlo para que todas sus Operaciones fueran Online, es decir, eso que ahora damos por sentado: que toda operación que hagamos en un banco cualquiera se refleje inmediatamente en nuestra posición en los ordenadores centrales. No muchos bancos del mundo tenían Sistemas Online al comienzo de la década de los ochenta... y prácticamente todos ellos, al final.

En este capítulo describiré someramente el método de trabajo que usábamos durante estos años, método que, a su vez, cambió radicalmente a mediados y finales de la década, con la adopción generalizada del PC en la empresa, lo que contaré en próximos capítulos.

En primer lugar, unas líneas (bueno, *muchas* líneas, que yo no sé resumir) para contaros **cómo trabajaban Bancos y Cajas de Ahorro a finales de los años setenta y principios de los ochenta.**

Todos fueron completando más o menos su Sistema de Información a lo largo de la década, y todas las Aplicaciones importantes estaban funcionando, pero pocos tenían alguna parte online.

**Cada oficina llevaba exclusivamente la posición de sus clientes.** Si querías hacer una operación en una oficina que no fuera la tuya... lo llevabas claro. Esta posición se actualizaba en unas fichas de cartón bastante grandes, de unos cuarenta por treinta centímetros aproximadamente, creo recordar, que ya no se actualizaban a mano (bueno, en algunas entidades, todavía sí), sino con unas máquinas específicas para ello, de las que había una o dos por oficina.

Estas aparatosas máquinas imprimían los movimientos y la posición resultante en la ficha y, además, grababan la información del saldo de la cuenta en una pequeña banda magnética que la propia ficha de cartón llevaba a un lado, donde se guardaban esos datos.

No he encontrado ninguna imagen en parte alguna, pero imaginaros una especie de “*Libreta de Ahorro Gigante*” con una sola página y su banda magnética al dorso... pues algo parecido. En realidad eran *dos* cartones, separados con papel de calco para pasar lo impreso a la copia. Cuando un cliente pedía los movimientos... se cortaba con unas tijeras el trocito de la copia donde estaban reflejados los apuntes pedidos y se le entregaba el pedazo. Con suerte, hasta se podía descifrar sin necesidad de un criptógrafo lo que había allí escrito.

Comenzó a ser corriente en la época que los bancos tuvieran en las oficinas, al menos en las más importantes, ciertos mini-ordenadores especializados en las Aplicaciones más cruciales: Cuentas Personales, sobre todo. Y cuando digo “*especializados*” me refiero a que estaban programados con el software adecuado para esa única aplicación. De muy escasa capacidad (16, quizá 32 Kb), discos flexibles de algunos cientos de Kb (de esos grandes de 8 pulgadas, no los más modernos de 5,25), y programados en lenguajes específicos de cada uno, generalmente ensambladores endemoniados, al menos para mi gusto, servían para mantener la posición de los pocos miles de cuentas que podía tener una



oficina normal de los años setenta.

Mantenían el saldo, validaban y anotaban los movimientos (ingresos, cheques, recibos de caja, etc.), eran capaces de imprimir un extracto a petición... en fin, podían resolver el 95% de la operativa normal de la oficina con sus cuentas “casi” online; al menos, desde el punto de vista del cliente, efectivamente lo era.

En realidad supusieron un enorme avance cualitativo: antes, para saber si cierta cuenta tenía o no saldo para responder a un pago, y siempre en la propia oficina donde la cuenta estaba abierta, el Cajero tenía que levantarse, ir a la gaveta de las fichas (las de cartón) y mirar qué saldo había allí reflejado; como los movimientos no se anotaban inmediatamente en la ficha, sino que se “pasaban” generalmente al final del día, podía haber desagradables sorpresas por haber pagado varios cargos contra el mismo saldo y otros inconvenientes similares.

Con estos minis **se podía consultar el saldo y ¡actualizarlo! inmediatamente** y sin que el Cajero se levantase de la silla... bueno, no para mirar el saldo, pero sí para comprobar la firma; lo que pasaba es que el cajero conocía al 90% o más de los clientes de la oficina, así que no necesitaba casi levantarse para este menester.

Los responsables de los Bancos se dieron cuenta palpable de que tener los saldos online era realmente una ventaja competitiva. Un gran avance, en definitiva. Además, *al final del día eran capaces de transmitir por vía telefónica los movimientos a la central*. Casi siempre utilizando la Red Conmutada, o sea, el teléfono normal: una llamada de teléfono a uno específico en la central de Proceso de Datos y, una vez establecida la comunicación, transmisión de los datos, por lo general vía tonos audibles, a una velocidad que rara vez pasaba de 1200 bps (o sea, 150 caracteres por segundo).

Y todo eso, sin ningún tipo de codificación o criptografía, que yo sepa: nadie estaba entonces preocupado por el robo de información por la línea: *ni siquiera imaginábamos que semejante cosa pudiera llegar a hacerse...*



En la imagen podéis disfrutar de un anuncio de un módem de 1200 bps de 1972. ¡*Vaya tela!*, decir que un módem puede ser *sexy*... sobre todo ¡ese módem...!

Una vez acabado el envío, en Proceso de Datos (realmente, en Grabación) se grababa el fichero recibido en una cinta magnética y ésta se enviaba al Ordenador Central para ser procesada.

Este sistema permitía tratar los movimientos del día en el *pase nocturno* del propio día, lo que representaba un gran avance sobre el Sistema tradicional: apuntar los movimientos del día en un formulario (por algo a los movimientos bancarios se los denomina "*apuntes*"), enviarlo por valija a la central mediante una especie de mensajeros que había entonces para el envío urgente de este tipo de documentos, pues UPS y similares no habían sido inventadas todavía, y allí ser grabado en el Departamento de Grabación para generar el fichero en cinta con los movimientos... de dos o tres días atrás.

Por otra parte, había el problema de que estos miniordenadores eran tan caros y su mantenimiento tan costoso, sobre todo en las oficinas que no estuvieran situadas en Capitales de Provincia, que no era factible instalar este Sistema en todas las oficinas, sólo en las realmente grandes que justificaran su coste.

En este estado de cosas, era evidente que el proceso online de *toda la información* reportaría grandes ventajas a la banca (y también al resto de grandes empresas), y además a un coste mucho más razonable que por el procedimiento manual explicado, así que todas las entidades se embarcaron por aquellos años en desarrollar su Sistema de Información online. **Cada una, el suyo**, desde luego. La mayoría, usando mainframes de IBM (muchas ya tenían ordenadores de esta marca, otras aprovecharon el momento para cambiar), y muchas de ellas, miniordenadores especializados en banca (terminales financieros) de la propia IBM, aunque también los había de otras marcas y se integraban bastante bien.

Veamos qué características tenía el Sistema de ese (mi) Banco.

Bien. El **Ordenador central** era un **mainframe de IBM**. Quedó claro antes, ¿no?

En mi caso, o más bien, en el del Banco donde trabajaba por entonces, un **IBM 3081**, pero según fuera la fecha de la compra, sería un modelo u otro (antes de eso, 370, 3031, 3033, 4341, 4381 y, más adelante, el resto de la gama 308x, 3090, 390, etc), de menor o mayor potencia en cada caso. En realidad, el que sea un modelo u otro no importa mucho, puesto que todos ellos son compatibles hacia adelante y lo que funciona en uno lo hace también sin cambio alguno en todos los modelos posteriores más potentes, incluso en nuestros días.

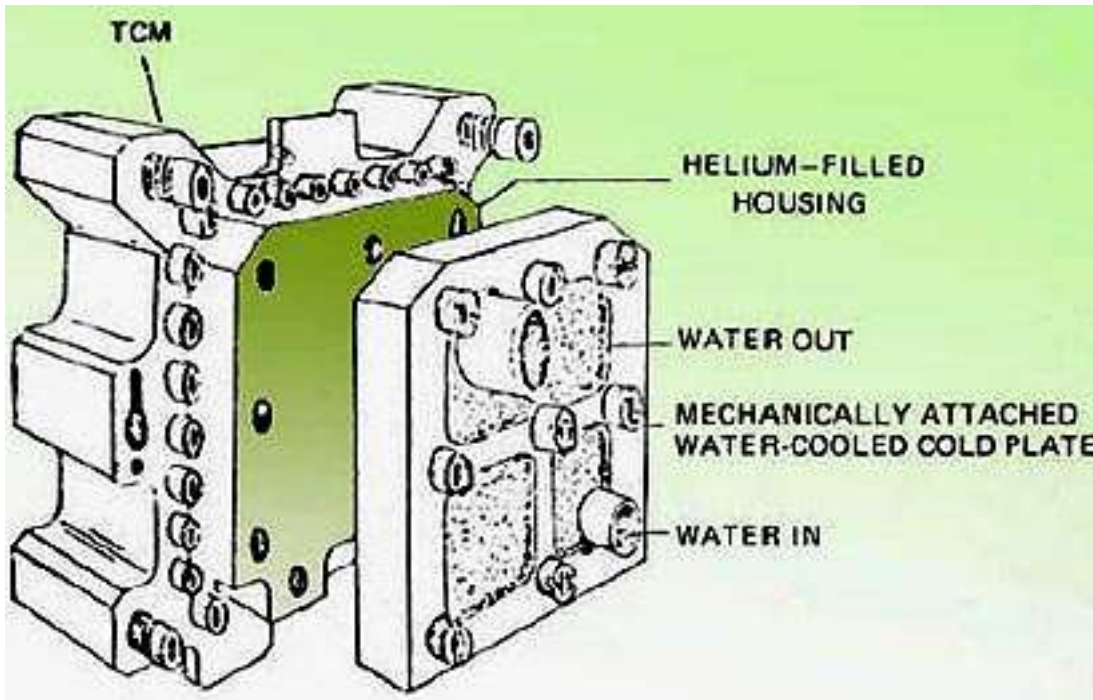
Esta característica facilita enormemente el cambio de un ordenador obsoleto a otro más moderno, al tener la seguridad de que todo el software, sobre todo, *el que tú has escrito*, funcionará sin problemas en la nueva máquina.

Aquel 3081, culmen de la miniaturización en 1980, tenía dos procesadores y 16 Mb de memoria, muchos canales, no recuerdo cuántos, de entrada/salida, varios Gb de espacio en discos magnéticos, cuatro unidades de cinta magnética y una velocidad de proceso nunca vista: sus ciclos de reloj eran de 26 nanosegundos, lo que quiere decir que **la velocidad del procesador era de cerca de 40 Mhz**... ¡ya en 1980!

Sin embargo, tal grado de miniaturización en 1980 tenía sus inconvenientes: los circuitos lógicos despedían mucho calor. Pero *mucho calor*. Tanto (unos 300W por módulo, cuyo tamaño era más o menos de 6x6 centímetros) que, sencillamente, era imposible que fueran viables, pues el calor disipado fundiría literalmente los circuitos... ¡incluso provocaría un incendio! Para solucionar este serio problema, IBM introdujo en su gama

308x (el primero de ellos fue el 3081) los **TCM's** (Thermal Conduction Modules), diseñados para disipar eficientemente el calor generado.

Todos los circuitos del sistema, como procesadores, controladores, canales de entrada/salida, etc., estaban encapsulados en los TCM's, cerrados herméticamente y *reellenos de helio* (no, líquido no: helio *gaseoso*), el mejor conductor de calor que encontraron, y el propio TCM estaba a su vez externamente refrigerado por agua enfriada, que circulaba constantemente accionada por bombas hidráulicas, casi como si se tratara una diminuta central nuclear.



En la ilustración anterior, un diagrama del dichoso TCM, donde se indica el contenido de helio, las entradas y salidas de agua... Curiosos, estos ordenadores refrigerados por agua. Nosotros siempre hacíamos el chiste fácil, llamando a la Sala y preguntando por el *Fontanero* de Sistemas... En estos tiempos parece casi una extravagancia, pero en aquel tiempo la tecnología no daba para más.

Y sin embargo esta tecnología, un tanto aparatosa, funcionaba muy bien. Eran ordenadores fiables, que rara vez tenían un problema, y cuando lo tenían, la reparación era muy rápida: se cambiaba el TCM completo donde radicaba el problema por otro idéntico, y listo.

Por cierto, ésta es la primera vez que el mantenimiento de ordenadores se basó en *sustituir la pieza averiada por otra de repuesto*, en lugar de reparar la pieza rota, polímetro y soldador en ristre, que era lo normal hasta entonces. Esta técnica de mantenimiento “por recambio” y no “por reparación” obviamente se impuso y dura hasta nuestros días.

El sucesor del IBM 308x, el IBM 3090, también se mantenía de esta manera, aunque estaba ya refrigerado por aire. Por aire frío. *Muy frío*. El ordenador era muy potente, cierto. Pero, como siempre ocurre, lo que lo hacía de verdad grande a un ordenador era su software. Veamos:

El **Sistema Operativo** era, en casi todos los casos, **MVS**. Entonces se llamaba

MVS a secas, años más tarde se llamó MVS/XA -por cierto, ésta fue la primera versión que permitió “romper” la línea de los dieciséis MB en un solo trabajo, al comenzar a utilizar 31 bits para direccionamiento, en vez de los 24 bits que se usaban hasta entonces-, años después MVS/ESA, luego OS/390 y ahora se llama z/OS, que direcciona ya con 64 bits... pero mayormente sigue siendo el mismo, muy evolucionado, eso sí.

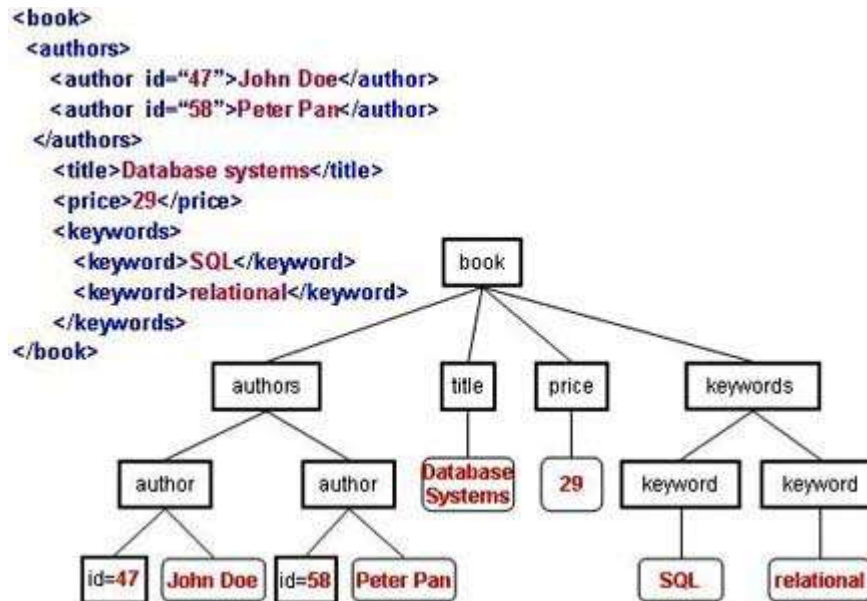
Si queréis saber más sobre cómo se desarrolló (que yo sepa, de la nada) este Sistema Operativo, nada como leer **“El Hombre-Mes Mítico”** (“The Mythical Man-Month”), de Fred Brooks, director del proyecto de desarrollo del OS/360, que más adelante se acabó convirtiendo en el MVS. Éste es un libro de culto entre los informáticos, pues en él Brooks habla de sus experiencias dirigiendo el mayor proyecto de desarrollo de software jamás realizado hasta entonces (más de 3.000 años hombre de esfuerzo). Si la frase *“Añadir recursos a un proyecto retrasado sólo conseguirá retrasarlo más”* os suena de algo, pues es de Brooks. Y el libro se publicó ya en 1975...

Como **Gestor de Bases de Datos, se usaba IMS/DB**, también conocido como DL/1, aunque en realidad DL/1 (por “Data Language/1”) era el Lenguaje de manipulación de la información en la Base de Datos. IMS/DB es (aún quedan por ahí aplicaciones escritas en DL/1) una Base de Datos jerárquica, es decir, su diseño toma forma arborescente, donde cada registro (Segmento, en su terminología) puede tener “*padres*” (de los que hereda sus claves), “*hijos*” (a los que cede sus claves) o “*hermanos*” (otros hijos del mismo padre, que están físicamente ordenados entre sí por algún criterio o clave).

Esta Base de Datos no es sencilla ni de diseñar ni de programar correctamente, pero una vez hecho tiene un rendimiento extraordinario, no sobrepasado aún hoy en día, y fue capaz de sacar un excelente partido a los ordenadores de la época, de limitadísimas capacidades si lo comparamos con *cualquier* ordenador actual, incluyendo el que tú, lector, tienes encima o debajo de la mesa.

En el mercado había alguna otra Base de Datos por entonces, todas ellas jerárquicas o en red, pero IMS fue la que mayor éxito comercial tuvo, sobre todo en los ordenadores de IBM, por motivos obvios.

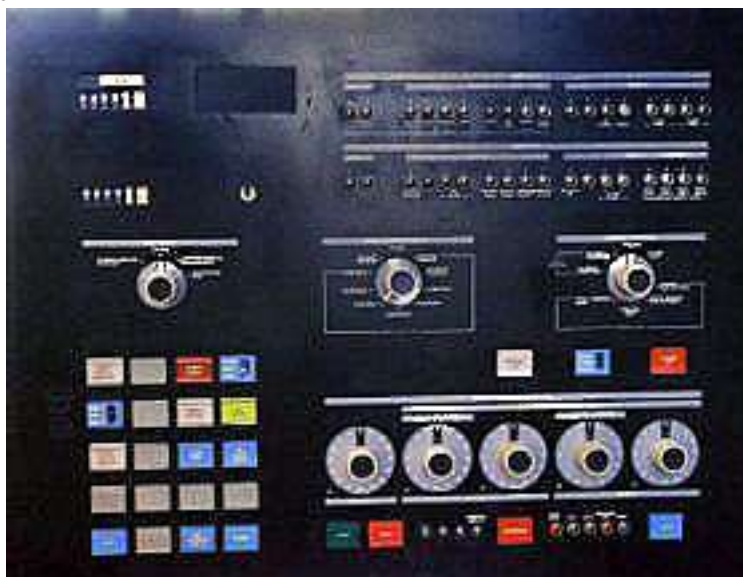
Por cierto, sé que la mayoría de vosotros, queridos lectores, estáis convencidos de que las Bases de Datos Jerárquicas han pasado a la historia, fagocitadas por las hoy omnipresentes Bases de Datos Relacionales, pero quizá cambiéis vuestra opinión cuando penséis, por ejemplo, que el hoy tan usado lenguaje XML es, en realidad, puramente jerárquico. Y si no, observad la ilustración para comprobarlo fehacientemente:



Para almacenar información descrita en XML (información *en serio*, quiero decir) es mucho más eficaz utilizar una Base de Datos Jerárquica que no una Base de Datos Relacional, donde para plasmar la Jerarquía se precisan tablas de relación entre tablas, lo que hace la solución mucho menos eficaz: para recuperar una rama completa del árbol es preciso hacer join tras join... y los joins relacionales no suelen ir muy rápido, precisamente. Cuando tenemos miles de registros no importa demasiado, pero si tenemos almacenados decenas o cientos de millones, entonces **sí que importa**.

En cuanto a las **comunicaciones con las oficinas**, se hacían mediante una IBM 3705, Unidad de Control de Comunicaciones, donde funcionaba el NCP (Programa de Control de Red), que permitía llevar el control de las comunicaciones con los terminales liberando al mainframe (y sobre todo, a su software) de llevar ese control.

En la ilustración, el panel de control de una unidad 3705, lleno de diales y botones y lucecitas...



IBM SNA era el protocolo de comunicaciones, que permite comunicaciones

síncronas bajo SDLC, con bajo overhead de comunicaciones.

Ni de lejos es tan potente como nuestro ubicuo TCP/IP... pero era perfecto para las líneas de la época: punto a punto con cada oficina, de 4.800 ó 9.600 bps, como máximo. También las comunicaciones han mejorado *bastante* desde entonces.

Toda la gestión de sesión la hacía otro programa básico dentro del software de mainframes: el VTAM, esta vez residente en el mainframe, no en las 3705, que era el que se encargaba de “repartir juego” entre las diferentes aplicaciones destinatarias: el IMS, el CICS, el TSO, o incluso las aplicaciones RJE, las que permitían imprimir listados obtenidos en el host por impresoras que estaban físicamente en oficinas, ahorrando una fortuna en transporte de los documentos, que no en papel, por cierto, pues de todos modos los listados se seguían imprimiendo, pero en las oficinas.

El **Gestor Transaccional** era **IMS/DC**. Otra posibilidad de la propia IBM, de origen más antiguo, era CICS, aunque el diseño del CICS estaba más orientado a pequeños sistemas transaccionales, mientras que el IMS lo estaba a los verdaderamente grandes... aunque en la actualidad, tras treinta años de mejoras, ambos tienen capacidades similares.

Un Gestor Transaccional es el Subsistema encargado de recibir las transacciones que se producen en las oficinas y garantizar su proceso correcto. Transacciones son las operaciones elementales: un ingreso de efectivo en una cuenta, un pago por cheque, un cambio de domicilio, una consulta de movimientos... las miles de posibles operaciones que realizan los bancos cada día son todas ellas transacciones. El Gestor Transaccional tiene que:

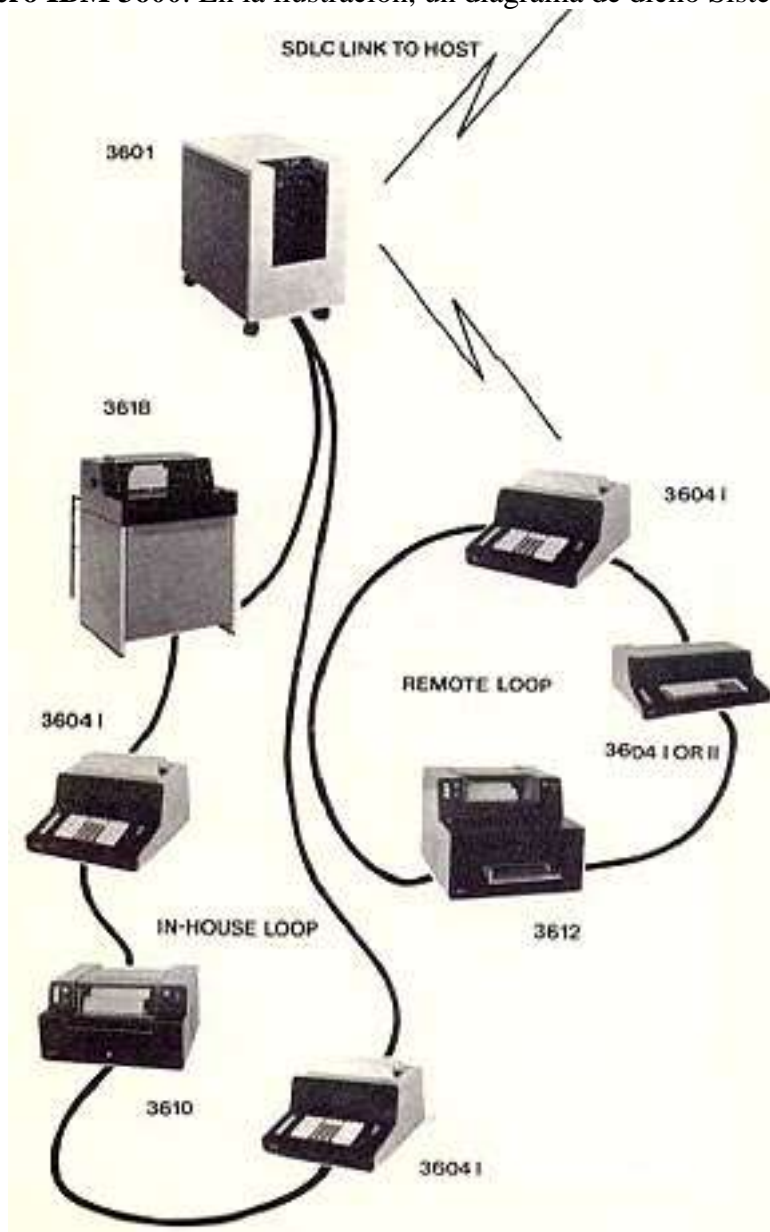
- 1 Reconocer de qué transacción se trata para asociarla al programa que la resuelve.
- 2 Gestionar las colas: en momentos de gran tráfico, es posible que se reciban más transacciones por segundo de las que se puede despachar, por lo que debe poner las transacciones individuales en su respectiva cola y gestionarlas.
- 3 Invocar (o sea: cargar en memoria y ejecutar), cuando le toque según su prioridad, al programa que leerá los datos de la transacción concreta de la cola para procesarlos.
- 4 Gestionar los accesos que el programa haga a las Bases de Datos, teniendo en cuenta posibles concurrencias, bloqueos, fallos de programa, etc.
- 5 Cuando el programa termina con el proceso de la transacción, enviar la respuesta al terminal de origen, vía las correspondientes colas de salida, con las mismas funciones que las de entrada. O también insertar en la cola de entrada un mensaje para arrancar otra u otras transacciones para que hagan alguna función (avisar a un supervisor, por ejemplo), o que terminen el proceso.
- 6 Y cuando el programa falla, porque en ocasiones, aunque no os lo creáis, los programas *fallan*, recuperar todos los cambios que éste haya podido realizar para dejar los datos como si nada hubiese pasado.

Y todo esto, ejecutando muchas transacciones simultáneamente. Decenas, o centenares, o miles de ellas por segundo. Con un tiempo de respuesta normalmente por debajo del segundo. Al cabo del día serán varios millones de transacciones las que habrá tratado un solo IMS/DC... sin pestañear. Y sin caerse. Aunque casquen los programas que dan servicio a las transacciones, por el motivo que sea (que **eso sí que pasa**, que al fin y al cabo los programamos *nosotros*), el IMS en sí ni se inmuta. Un programa bien hecho, el



IMS/DC.

Por fin, **en las Oficinas del Banco** había efectivamente miniordenadores específicos para la gestión de las transacciones bancarias: me refiero al así llamado **sistema financiero IBM 3600**. En la ilustración, un diagrama de dicho Sistema.



Se trataba de un servidor, de gran tamaño físico para lo que ahora estamos acostumbrados, conectado por *cable coaxial* con los terminales bancarios de la oficina y, por *línea punto a punto* (entonces, de 9.600 bps como máximo, o, lo que es lo mismo, 1.200 caracteres por segundo), a la central... o a otras oficinas más pequeñas, a las que daba también servicio, con el correspondiente ahorro de coste.

Por la línea enviaba las transacciones (las operaciones que se ejecutan) y recibía las respuestas ("*Reacciones*"); por los terminales realizaba la captura de las transacciones, una

cierta validación de forma (que no falten campos obligatorios, que las fechas sean correctas y poco más) e imprimía los documentos bancarios y, ¡oh maravilla!, también la hoja de fondo.

Esto último puede parecer una tontería hoy en día, pero eran capaces de imprimir en papel continuo, en la hoja de fondo, toda la información de control para cuadros, auditorías, etc., y validar por la impresora de documentos los resguardos para los clientes.

Para realizar esa validación de documentos, lo que tenía era en realidad otra impresora diferente que, ¡en 1980!, no era de papel continuo, sino que se “come” el documento, lo imprime y luego lo devuelve: tecnología punta de la época, una característica novedosa que aportaron esos sistemas especializados que, como podéis suponer, se programaban en su propio Ensamblador endiablado.

Bien, ahora que he descrito sucintamente el *paisaje*, voy a contar a qué se dedicaba el *paisanaje*, es decir, **cómo diseñábamos, programábamos, probábamos y poníamos en Producción las Aplicaciones en este entorno** novedoso para mí, aunque no tanto, pues en la Carrera habíamos estudiado bastantes de las características de estos ordenadores, incluso hecho algún programa para él, y a mí no me resultó desconocido.

¿He dicho ya que *todo el software se hacía ex profeso para cada empresa*? Lo dije hablando de los años setenta y lo repito al hablar de los ochenta. Pero lo que sí **estaba cambiando rápidamente era la forma de diseñar y escribir el software**. No sólo los medios técnicos eran mucho más avanzados, sino que **el propio método que seguíamos era más avanzado**.

Todo el **Sistema de Información** fue diseñado completamente por profesionales del Banco ayudados por Técnicos de IBM con experiencia en banca y programado por programadores de plantilla del Banco, todo a lo largo de tres o cuatro años. El Plan de Formación previo fue exhaustivo, pues todo el personal de Proceso de Datos del Banco, así como las nuevas incorporaciones como yo, pasamos varios meses aprendiéndonos de arriba abajo el nuevo entorno técnico, muy potente, cierto, pero también muy complicado. El método de trabajo que seguíamos entonces era, *aproximadamente*, el siguiente:

Los Analistas Funcionales (ya no necesariamente “Jefes de Proyecto”), casi todos del Departamento de Organización y Métodos, escribieron el **Análisis Funcional** de las distintas Aplicaciones: Cuentas Personales, Contabilidad, Valores, etc. No había directrices claras sobre cómo escribir estos documentos, así que cada analista lo escribía como sabía y como podía. Ahora lo llamaríamos más bien un “*Análisis de Requerimientos*”. A veces eran muy detallados en la descripción de la operación elemental y a veces terriblemente ambiguos... claro que *ahora sigue pasando lo mismo*.

La colección de todos estos “Análisis Funcionales”, junto con las especificaciones de la solución técnica, formaba el “**Plan de Sistemas**”, piedra angular a la que todos nos referíamos cuando había discrepancias. Que las había.

Terminado el Análisis Funcional/de Requerimientos, se realizaba el **Diseño Técnico** de la Aplicación. Esto consistía en las siguientes tareas:

**1 Diseñar las Bases de Datos IMS** de la Aplicación. Un punto crítico, porque un fallo en el diseño inicial podría costar muchísimo tiempo y esfuerzo (y por tanto, dinero) en corregirlo, una vez terminados los programas.





En la ilustración, el diseño de una Base de Datos IMS de Cuentas de un Banco. El diseño real sería *ligeramente* más complicado...

Gastábamos muchísima saliva en interminables reuniones para determinar, entre todos, el mejor diseño posible teniendo en cuenta los requerimientos, el número de transacciones esperadas, etc. Y tengo que decir que al final casi siempre llegábamos a un consenso de diseño que, en general, era el menos malo de los posibles, porque ya sabemos que el mejor, lo que se dice *el mejor*, no existe.

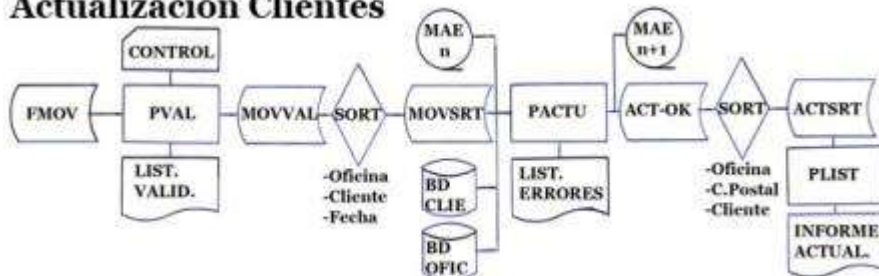
**2 Diseñar las transacciones**, es decir, qué datos deben viajar entre el terminal financiero (el famoso IBM 3600) y el Ordenador Central, para minimizar el tráfico, cosa que era crítica debido a la bajísima velocidad de transmisión, pero sin que falte ningún dato necesario.

**3 Repartir las funciones de la transacción:** qué debía hacerse en el terminal financiero, más bien poco, debido a su limitada capacidad, y qué en el mainframe, casi todo, en realidad.

**4 Diseñar el batch.** Muy importante, porque las transacciones capturan la información, la validan y toman decisiones inmediatas, pero toda la consolidación se hacía y, en buena parte, se sigue haciendo hoy en día en batch, en un proceso por lotes. Muchísimos procesos son netamente batch. Liquidar cuentas, Emitir Recibos, Imprimir Extractos Mensuales, Hacer el Balance de Contabilidad, Abonar Dividendos, etc., son procesos costosos que necesitan acceder a muchísima información y donde no hay nadie esperando respuesta al otro lado del terminal. Todo esto se hace normalmente en batch.

Podéis ver a continuación un diagrama básico del proceso batch de Actualización de un Fichero Maestro cualquiera, verbigracia, el de Clientes.

#### Actualización Clientes



La **ventana batch** (*lugar* donde se ejecutan la mayor parte de los procesos batch) lo explicaré en detalle en un capítulo exclusivo para ella más adelante, puesto que es un concepto muy importante y muy desconocido para muchos de los más jóvenes.

**5 Diseñar los módulos comunes.** Son piezas de software reutilizables en diversas partes de una Aplicación, que se aíslan de antemano y se programan una única vez, asegurándose de que hacen perfectamente la función definida y con la máxima eficiencia posible.

Los había genéricos, que utilizaban todas las Aplicaciones, como, por ejemplo, dadas dos fechas, calcular la diferencia en días entre ellas, dado que esto se usa muchas veces en los procesos bancarios, y otros específicos de cada Aplicación, como el módulo de Contabilización de Apuntes, que sabía a qué cuenta contable se refería cada tipo de

movimiento y realizaba correctamente y sin ayuda, el apunte en la Base de Datos de Contabilidad... de hecho, con el tiempo, era *el único* que sabía dónde demonios se contabilizaban las cosas. O sea, que lo de la reutilización no lo han inventado con los omnipresentes Objetos de hoy en día... ya lo habíamos inventado nosotros muchos años antes.

## **6 Diseñar los programas de interfase con las Aplicaciones Antiguas.**

*Importantísimo:* Naturalmente, las Aplicaciones no se arrancaron todas a la vez; de hecho cada Aplicación tampoco se arrancaba en todas las oficinas a la vez, técnica que, por lo que yo sé, se sigue usando hoy en día. Entonces, al arrancar Cuentas Corrientes en un par de oficinas, deberían sus datos integrarse con los del resto de Oficinas, que seguían trabajando con las Aplicaciones Viejas hasta que todas las oficinas estuvieran migradas a la nueva Aplicación. Y, en cualquier caso, esta Aplicación de Cuentas Corrientes debía comunicarse con las de Contabilidad, Valores, etc., que seguían siendo las antiguas, hasta que pudieran ser a su vez sustituidas, meses o años después. Pregunta retórica a aquellos lectores que estéis estudiando informática: *¿Alguien os ha contado estas cosas?* Si la respuesta es **Sí**, podéis felicitaros, a vosotros mismos y a vuestros profesores.

Una vez terminado este Diseño Técnico, fundamental para que todo vaya luego bien y, sobre todo, para que la Aplicación resultante sea “mantenible” de cara al futuro, había que **realizar las especificaciones de los programas individuales**, para su programación. Éste fue, mayormente el trabajo que comencé haciendo allí, como “Analista Orgánico”, participando también cuando hacía falta en el Diseño Técnico de la etapa anterior. Pero si había que programar, se programaba, y si había que hacer un Funcional, se hacía. ¿He dicho ya que el objetivo de todo el equipo era sacar el trabajo adelante y que todo el que podía colaborar a hacer algo bien, lo hacía, sin preocuparse de las categorías profesionales y otras zarandajas?... Y además, los fines de semana nos íbamos todos a jugar al fútbol (no, las chicas no iban, ni siquiera a animar) y *casi* no nos dábamos patadas entre nosotros...

Para hacer el Análisis Orgánico de los Programas, tanto los online (transacciones) como los programas batch, realizábamos los “**Cuadernos de Carga**” donde el Analista describía las principales funciones que debía realizar el programa. **A lápiz**, oiga, para poder borrar. Y usando la misma Plantilla que usábamos años atrás para pintar los diagramas de los programas en los Cuadernos de Carga.

Preparamos una serie de hojas standard preimpresas, donde dibujábamos con la plantilla el diagrama del programa (Bases de Datos a las que accedía, Ficheros que leía/escribía, Impresos que generaba, etc).

Después, en *román paladino*, describíamos con cierto detalle lo que tenía que hacer el programa de marras: Que si leer los datos de la transacción de entrada; que si validar de tal o cuál manera los datos; que si comprobar de tal otra si había saldo o no en la cuenta; qué hacer si no había saldo o si la cuenta estaba bloqueada o intervenida; cómo actualizar saldos y movimientos, qué responder y de qué manera a la oficina... en una palabra, esas indicaciones que permitían al programador escribir el programa y que éste *casi* hiciera todo lo que debía hacer...

Nos convertimos en virtuosos del *copiar-y-pegar*, pero literalmente, no como ahora, que es todo virtual: hacíamos una fotocopia de la página que había que copiar, recortábamos la parte que nos interesaba, la corregíamos si era preciso con el Tipp-ex, o sea, *blanquillo*, y luego escribiendo encima, y pegábamos por fin el resultante en la hoja definitiva con goma arábica. ¡Un auténtico lujo!

Las transacciones online sólo podían acceder a las Bases de Datos IMS, que obligatoriamente debían residir en disco, pero los ficheros Maestros seguían estando mayormente en cinta magnética, que se trataban en baterías de decenas o veintenas de Unidades de Cinta. Los discos seguían sin tener capacidad suficiente como para guardar allá toda la información necesaria. Por lo tanto, para actualizar estos ficheros en cinta, mediante cadenas batch compuestas de una serie consecutiva y secuencial de programas tuyos y programas de utilidad, sobre todo el omnipresente Sort para ordenar ficheros, se utilizaba un tipo de proceso de actualización denominado “Padre-Hijo”, que ya expliqué en capítulos anteriores y que continúa siendo muy útil hoy en día, aunque desde luego casi nadie lo reconoce...

En el otro lado de la línea estaba la misma transacción, pero vista desde el terminal financiero. Había, pues, que hacer otro Cuaderno de Carga por otro Analista diferente y especializado en este Sistema Financiero, coordinándose con el del mainframe, para tratar la misma transacción, especificando aquí lo que debía hacer en el *front-end*, es decir, en la parte que “da la cara” ante el cliente: Definir qué campos son obligatorios y cuáles opcionales para esta transacción, cómo validarlos, cómo rellenar el mensaje para la central, enviarlo...(esperar respuesta)..., cómo procesar la información recibida del Ordenador Central, qué imprimir y dónde, qué hacer si hubo un error, etc.

Los programadores recibían los cuadernos de carga una vez terminados y comenzaban entonces su trabajo: **codificar, compilar y probar sus programas**. Inicialmente usábamos la misma técnica que describí previamente para los NCR Century 200:

**Uno:** Escribir en Hojas de Codificación el programa en el lenguaje adecuado (Endemoniado Assembler en el terminal financiero, Cobol en el Ordenador Central),

**Dos:** Enviarlos a perforar,

**Tres:** Cuando los devolvían perforados se compilaban, el listado de la compilación se enviaba al programador, etc., etc... en fin, no sigo: el mismo vetusto procedimiento que ya conté en el capítulo dedicado al método de trabajo de los setenta.

Lo que sí cambió radicalmente fue *el modo en que se diseñaban y escribían los programas*: Desde el primer programa que hicimos, utilizamos las técnicas de **Programación Estructurada**. Se acabaron los organigramas chapuceros, se acabó el dichoso código spaghetti, se acabaron los GOTO's indiscriminados a cualquier parte del programa... Se acabó, en definitiva, programar como a cada uno buenamente se le ocurría: se acabó la programación artesanal de décadas anteriores, para entrar en una programación más industrial, más organizada y previsible. Bueno, más o menos. Los programas resultaban más eficaces en ejecución, se tardaba menos en codificarlos y probarlos y, sobre todo, *sobre todo*, eran mucho más sencillos de mantener.

En la actualidad “*se da por supuesto*” que los informáticos, y los aspirantes también, saben las normas básicas de este paradigma de diseño, pero apenas se enseña (y conoce) de verdad por los profesionales de nuevo cuño... y los viejos lo hemos olvidado. El próximo capítulo estará dedicado a explicar esta técnica, la Programación Estructurada, con algún detalle, intentando contaros cómo empezó todo, cómo fuimos *cambiando el chip*, qué se fue haciendo para mejorar la programación, qué normas se ponían para obligarnos y cómo nos las saltábamos... Pero por ahora, sigamos programando en los ochenta.

Al principio seguimos utilizando **fichas perforadas** (al fin y al cabo, la costumbre

es la costumbre), pero las cosas cambiaron muy rápidamente, porque una de las características más importantes del MVS era su estupendo **TSO**, subsistema de tiempo compartido que permitía a diferentes usuarios, usando una interfaz de línea de comandos por pantalla, ¡no en ficha perforada, sino **tecleando en pantalla!**, una IBM 3270, acceder simultáneamente como si todo el mainframe fuera sólo suyo... cosa que no era cierta, claro, pero lo parecía.

O sea, que además de buenos programadores, nos convertimos también en malos mecanógrafos... y así seguimos.

Rápidamente se instaló también, sobre el TSO, el **ISPF** que permitía, ahora sí, un acceso mediante pantalla completa y un magnífico editor de programas (para qué mentir: el mejor que yo conozco... y mira que conozco), que revolucionó la informática y la forma de trabajar, sobre todo de programadores y operadores, pues nosotros, pobres Analistas, seguíamos escribiendo a lápiz páginas y páginas como tontos.



Esas pantallas 3270 como la de la imagen eran monocromas (de fósforo, en verde o en amarillo, según modelo), dedicadas, conectadas con cable coaxial al concentrador de comunicaciones que, en definitiva, era quien las conectaba con la IBM 3705, de veinticuatro líneas de ochenta caracteres, y desde luego que su interfaz no era gráfica, sino de caracteres (y sólo las mayúsculas) y a base de teclas de función (PF8, adelante; PF7, atrás; PF3, grabar y salir; PF5, buscar, y así... Y PF1 era “Help”, pero *nadie pulsaba esa teclita...* ¡Somos españoles, qué pasa! ...y preferimos gastar una hora intentando averiguar por ensayo y error cómo funcionan las cosas, antes que leer un manual... y en inglés, para más *inri*).

O sea, ahora, una vez leído por primera vez el taco de fichas, el programa quedaba almacenado en una librería de programas fuente (cada programador tenía la suya propia, que gestionaba a su conveniencia, almacenada permanentemente en disco, aunque todos podíamos acceder en caso de necesidad a las librerías de los demás), y entonces las modificaciones al programa para compilarlo hasta ponerlo a cero errores, o para conseguir que funcionara durante las pruebas, se hacían usando el **editor del ISPF** y mediante pantalla.

Lo que pasaba al principio es que estas pantallas eran caras (¡y grandes!... no, en realidad *la carcasa* era grande, la pantalla en sí era enana, de catorce pulgadas o quizá menos) y el ordenador tampoco tenía capacidad para aguantar muchas regiones de tiempo compartido simultáneamente, así que había pocas pantallas (como media docena, recuerdo) en una Sala de Reuniones (*ex-Sala de Reuniones*, ahora “*Sala de Pantallas*”).

Los programadores iban allá con sus listados, se ponían en una pantalla que estuviera libre, **editaban su programa, lo modificaban, lo compilaban, lo volvían a modificar y compilar hasta ponerlo a cero errores y lo probaban hasta que funcionaba**

**correctamente...** ¡o hasta que se le acababa su tiempo!

Porque, claro, *había bastantes más programadores que pantallas*, que eran un bien escaso y bastante solicitado (*peleado*, diría yo), así que pronto quedó claro que había que ordenar el asunto mediante una lista de espera donde te apuntabas, y te correspondía una hora de uso de pantalla. Al acabar tu hora, venía el siguiente y te echaba. Sin más. Como en las Pistas de Tenis: cuando se acaba tu hora viene la siguiente pareja con sus raquetas y te echa... pues igual, pero sin raquetas.

Y si, cosa rara, terminabas lo que habías ido a hacer en menos de tu hora asignada, le cedías tu pantalla a alguna *programadora de buen ver*, a ver si caía algo... que nunca caía. Supongo que las programadoras se lo cederían a los *programadores de buen ver*, pero no tengo datos: nunca fui agraciado. ¡Qué jóvenes éramos, pardiez!

Jóvenes, sí... pero ¿he dicho ya que el inglés no era lo nuestro? Sí. El *Idioma Moderno* que se estudiaba en toda la educación primaria y secundaria (el Bachillerato Elemental y el Superior) era francés, así que la mayoría nos poníamos a trabajar sin saber casi ni el *to be*... y eso dio origen a muchas anécdotas...

Recuerdo en cierta ocasión a un compañero intentando seguir las instrucciones de un manual (en perfecto inglés, *of course*) y despotricando porque el bicho no funcionaba como debía, mejor dicho, como el manual decía que debía funcionar. Resulta que el manual pedía introducir “the actual address” y él estaba, naturalmente, introduciendo la dirección *actual*, que lo ponía muy clarito. Y no iba.

Ay, los dichosos falsos amigos... ¿a quién se le ocurriría que *actual* en inglés no signifique “actual”, como Dios manda? De todos modos, treinta años después, y cuando ya es obligatorio para todo hijo de vecino estudiar algo de inglés, aunque no sea mucho, en ocasiones todavía te encuentras al principio de una película o telefilm o lo que sea un cartelito avisador: “*Based on actual events*”, mientras que una voz en off te lo aclara: “Basada en hechos actuales...”, por si no te habías enterado bien. Se ve que hay cosas que no cambian.

De todos modos, la anécdota más graciosa que recuerdo al respecto tiene que ver no tanto con el propio inglés sino por *cómo lo pronunciamos*. En cierta ocasión el Banco en que trabajaba adquirió un producto que hacía algo muy importante que, para ser sinceros, no recuerdo. Igual tampoco era *tan* importante...

Para instalar el producto (en el mainframe, desde luego, a ver dónde si no), vino, provisto de unas cintas magnéticas, un alemán de Hannover que hablaba un inglés bastante bueno, y me correspondió a mí ayudarle para instalarlo debido a que yo era el que mejor inglés hablaba por allí. Imaginaos cómo sería el de mis compañeros... En cierto momento, ya conectados a la bestia vía el TSO y el ISPF, el caballero me dice que ahora necesita un “AIBISHINA” o algo así... Le miro con cara rara y le digo, en mi perfecto inglés...  
*¿Mandéee?*

El hombre me mira con cara más rara aún y me dice, esta vez silabeando: “AI-BI-SHI-NAH”. Pues no, la H del final como que no me dice nada, sigo sin saber qué rayos me pide. Así que le digo que lo siento, pero que no, que en la instalación no tenemos nada de eso. Me dice: “*It’s impossible*”, y yo le contesto, apesadumbrado, “*Well, that’s Spain... Everything is possible!*”.

Bueno, el hombre me mira con conmisericordia, se pone en la pantalla y comienza a teclear un JCL:

```
//JOB1 JOB MSGCLASS=A,etc (o sea, lo normal)
```

//STEP1 EXEC PGM=IEBGGENER...

¡Y de pronto se me enciende la luz! Y entonces le digo: “*Ahh! You mean I-E-BE-GE-NERR*”, señalando el nombre del programa que acababa de teclear: IEBGENER, uno de los programas de utilidad más usados en el mainframe, que sirve, sobre todo, para copiar ficheros de un lado para otro y que nosotros, como todo el mundo que trabaja con mainframes, usábamos quince veces diarias cada uno...

Y él me mira, señalando el mismo nombre: “Yes: AI-BI-SHI-NAH! (...lo que yo decía, pardillo) (esto último no lo dijo, que era muy educado, sólo lo pensó)”. ¡Con razón no podía entender que no lo usáramos nosotros! Y nos echamos unas risas a costa del malentendido... jamás se nos había ocurrido que tal cosa no se pronunciara en perfecto español y supongo que al de Hannover no se le habría ocurrido nunca que esas ocho letras se pudieran pronunciar de otro modo que en supuesto inglés... Cosas que pasaban en aquellos tiempos del cuplé.

Con el tiempo se fueron poniendo más pantallas, hasta diez o doce, luego quizá veinte, luego una cada dos programadores, hasta que no mucho después (sobre 1985 o así) todos los programadores (en realidad, todos los que accedían regularmente al Sistema) tenían ya en su mesa su propio *mamotreto*, digo pantalla, lo que agilizó mucho el proceso.

Rápidamente dejaron de perforarse los programas (con enorme alivio por parte de las pobres perforistas) y **los programas se escribían ya completos en la pantalla**, reutilizando partes de código similares de otros programas, inveterada costumbre de todo buen programador en Cobol. Cómo será el asunto que hay malas lenguas que afirman que no es verdad que haya en el mundo *millones de programas distintos escritos en Cobol*, sino más bien *millones de versiones distintas de un único programa*...

De hecho, creo que han localizado recientemente en Oklahoma al único programador que escribió una vez, cincuenta años ha, la sentencia “IDENTIFICATION DIVISION.”... ¡y desde entonces todo el mundo ha copiado esa línea!

Por fin, a mediados de los ochenta, llegó la hora de **mecanizar los Cuadernos de Carga**. No, *je, je*, todavía no se habían inventado las Herramientas CASE (o, si se habían inventado, no se habían puesto aún de moda, como pasó a finales de los ochenta y primeros noventa: en unos capítulos hablaré de ellas), la cosa era más rústica:

Tú, Analista Orgánico, seguías escribiendo en papel con tu lápiz, y después una mecanógrafa pasaba a máquina los Cuadernos de Carga en uno de los primeros sistemas ofimáticos que existieron, que eran unas máquinas “tamaño mesa” con un teclado, una pantalla y discos flexibles, que funcionaban básicamente como una máquina de escribir, con un sencillo editor de texto, sólo que lo escrito no se imprimía, sino que quedaba almacenado en el disco flexible y podía ser recuperado o modificado más adelante. ¡Adiós al *copiar-y-pegar-con-goma-arábica*!

En cualquier caso, cuando los programas por fin funcionaban (o, al menos, *lo parecía*), se ponían en **Producción**, por un procedimiento similar al descrito en el capítulo correspondiente al método de trabajo en los años 70, sólo que ahora Sistemas (los Técnicos de Sistemas, quiero decir, que creo que eran tres) copiaban tu programa fuente y el ejecutable y se lo llevaban a una librería del Sistema, donde ya los indocumentados de Desarrollo no podíamos modificarlo cuando lo necesitáramos o nos viniera en gana, como antes, sino sólo verlo o copiarlo. ¡Empezábamos ya a preocuparnos por cosas como la **Seguridad**! Si alguien nos cuenta entonces lo que hacemos *ahora* para garantizar la

Seguridad (no, sólo para *maximizarla*, porque garantizar, garantizar... en fin), nos caemos de la silla.

Cuando había que **modificar un programa** por el motivo que fuera, seguíamos diversos procedimientos dependiendo del motivo de cambio y, sobre todo, de la urgencia.

Si era un **cambio planificado**, los Analistas Orgánicos actualizábamos el Cuaderno de Carga (sí, generalmente más copy-paste con Tipp-Ex y goma arábica y, después, a pasarlo a máquina), y lo entregábamos al programador que lo iba a modificar, el mismo que lo escribió inicialmente siempre que se podía, que en aquellos años aún era posible; ahora, no. Este programador copiaba la versión “buena”, la que está en Producción en las *Librerías de Programas en Producción* (donde, insisto, los de Desarrollo sólo podíamos leer, no modificar nada), a su librería personal, localizaba el lugar donde tocar el código, lo tocaba, compilaba, probaba... y cuando estaba listo, daba orden a Producción para sustituir la versión del programa en Producción por la nueva.

Cuando las modificaciones afectaban a más de un programa, cosa habitual, había lógicamente que coordinar el paso a Producción de todos los programas afectados a la vez, en tal fecha, antes o después del proceso batch, etc.

Pero si había que **modificar un programa urgentemente** (casi siempre originado por un hermoso *casque*), pasábamos de documentar nada, e íbamos derechos al criminal causante de tamaña fechoría (¡cascar a las tres de la mañana, por favor!), a localizar el motivo del casque (mirando el listado con el volcado de memoria, claro, no había otra manera) y arreglar a todo trapo el programa para continuar la normal operación.

Para ello usábamos continuamente unos prontuarios que IBM tenía y distribuía y ahora me parece que ya no, con toda la información básica del Sistema que necesitabas para resolver el casque: la **IBM 370 System Reference Card** como la que tenéis en la ilustración siguiente. Por cierto, aunque diga “*IBM 370*”, seguía sirviendo para el IBM 3081, luego para el 3090, el 390, incluso para el z/Series actual.



En ella encontrabas a mano casi todo lo que necesitabas: todos los códigos máquina de operación, los de los caracteres, formatos de instrucción, contenido de la PSW (*Program Status Word*, lugar donde se almacena toda la información actual sobre la ejecución del programa), los códigos de error, etc.

Otra pregunta retórica, queridos lectores: ¿“**S0C7**” (*Ese-Cero-Ce-Siete*) os dice algo? ¿No? ...Pues es el código de casque más famoso de todos los tiempos y, aunque leído en inglés parece más impresionante, en realidad significa, exactamente: “*Chaval: has intentado hacer una operación aritmética con datos no numéricos*”. Conclusión: ¡A los corrales! Para otra vez, antes de operar, *valida* si los valores de los campos son realmente numéricos o no lo son, que para eso el Cobol te lo pone realmente fácil:

```
IF CAMPO-TAL-Y-CUAL NUMERIC realiza la operación  
ELSE algo va mal...
```

Aquellos fueron, desde luego, años realmente interesantes, quizá los mejores de mi vida... claro que por entonces tenía veintitantos años, ¡cómo no iban a serlo! Y sin embargo, pasada la mitad de la década volví a cambiar de Banco. Y esta vez, fue, más que por el Proyecto (que también), por el *vil metal*. Pero uno quería fundar una familia, y claro... Pero esta es otra historia... y nunca será contada.

La verdad es que no sólo el método de trabajo fue cambiando, sino también la forma en que se diseñaban los programas. Nuevos vientos estructurados venían de fuera y nos hicieron cambiar radicalmente de nuevo nuestro *modus vivendi*... o mejor, nuestro *modus programandi*.



## 7 – El Método de Programación Estructurada

En el capítulo anterior comenté, entre otras muchas cosas, que a principios de los ochenta se comenzó a utilizar en España de forma sistemática la nueva técnica de **Diseño Estructurado de Programas**. Esto no quiere decir que fuera inmediatamente adoptada por todas las empresas, no. Las empresas pioneras, el banco donde trabajaba entonces entre ellas, sí comenzaron a usar **Programación Estructurada** de forma generalizada, incluso, al cabo de poco tiempo, y visto lo bien que nos fue durante los primeros tiempos, *de forma obligatoria*. El diseño y codificación de los nuevos programas se hizo más rápido y con mayor fiabilidad, pero sobre todo se ganó muchísimo en el **mantenimiento de los programas** a lo largo de los años.

Pero la adopción de la nueva técnica no fue tampoco un camino de rosas. Siempre había algún virguero de la codificación tradicional que era capaz de, *GOTO p'arriba*, *GOTO p'abajo*, hacer el mismo programa más rápido y más eficiente que cualquiera, con o sin estructuras. Hubo bastante controversia, y no sólo en España, sobre la manera de codificar en los lenguajes de la época, todos ellos “*procedurales*”, desde luego, con o sin GOTOs, con o sin Inicio y Final obligatorio...

Antes de entrar en cómo nos fuimos adaptando al nuevo método de programación, debo, fiel a mi *estilo lenguaraz*, describir qué es la Programación Estructurada y daros unas pinceladas históricas.

Ya sabéis que en los cincuentas, sesentas y primeros setentas del siglo XX los informáticos de la época, en muchos casos ingenieros metidos a informáticos, programaban sus rutinas como buenamente sabían y podían. Fueron inventando las diversas maneras de programar conforme programaban, con lo que los más avisados (o afortunados, quién sabe) fueron encontrando maneras de solucionar problemas diversos sobre la marcha. Pero, al fin y a la postre, la forma aceptada de programar era mayormente a la “*mecagüendiez*”, traducción castiza del no menos castizo “*cada maestrillo tiene su librillo*”.

En el capítulo dedicado al método de trabajo de los setenta en Proceso de Datos ya incluí un organigrama añejo, copiado literalmente de la pizarra, donde, en Segundo de Carrera y con tiza blanca, un aguerrido profesor explicaba el método para actualizar ficheros indexados.

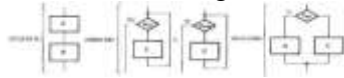
En una palabra, en los primeros setenta, *ésa* era toda la cera que ardía... Claro que, si nos damos cuenta, el problema no era *todavía* excesivamente importante: Había pocos ordenadores funcionando, no había muchos programas en Producción y, sobre todo, *estas eran muy pequeños*. No había otro remedio: con 8, 16 ó 32Kb de memoria no se podían hacer programas muy complejos ni queriendo, simplemente porque no cabían.

Sin embargo, alguien estaba ya dándose cuenta de que **esa no era la manera correcta de programar** de cara al futuro. Cuando el uso de las máquinas se fuera generalizando, éstas fueran cada vez más y más potentes y más y más programadores se fueran incorporando a la profesión para programar más y más aplicaciones de nuevo cuño, los programas *spaghetti* al uso no serían adecuados, debido no sólo a la dificultad de diseñarlos, programarlos y probarlos, sino, sobre todo, de mantenerlos. Efectivamente, conforme más y más aplicaciones se iban poniendo en marcha, más y más aplicaciones

había que mantener, modificar, cambiar... y entonces la programación “personal” constituía un problema serio, sobre todo cuando el autor original del programa ya no estaba en la empresa para desentrañar lo que ahí dentro había... y en muchas ocasiones, *ni así*.

En 1966, **Bohm y Jacopini** publicaron un artículo en el que demostraron formalmente que usando exclusivamente tres estructuras de control era posible programar cualquier función computable. Las tres estructuras eran y siguen siendo, claro está, **la secuencial, la repetitiva y la alternativa**.

Combinándolas recursivamente es factible realizar cualquier programa. Cualquiera, hasta el más extraño imaginable.



Estos son los diagramas de las tres estructuras básicas: Secuencia, Iteración y Selección.

**Una Secuencia** consiste en una serie de elementos que ocurren secuencialmente, uno detrás de otro, y siempre en el mismo orden. Esta serie de elementos puede estar vacía, es decir, no contener elemento alguno.

**Una Repetición** (o iteración) consiste en un único elemento que se repite una y otra vez, o ninguna, hasta que cierta condición sea satisfecha. O lo que es lo mismo, *de 0 a n veces*, mientras se satisfaga una cierta condición.

**Una alternativa** (o selección) consiste en dos elementos, de los que sólo ocurre uno u otro, dependiendo de cierta condición. Aunque en principio se definen sólo dos elementos para definir una selección, no hay ningún problema en permitir más de dos ramas, siempre que sólo ocurra una de ellas, dependiendo siempre de dicha condición (en realidad es el resultado de des-anidar varias selecciones anidadas).

No obstante su importancia para la ciencia de la computación, el artículo de Bohm y Jacopini fue poco menos que ignorado, ya que ellos se limitaron a demostrar matemáticamente que esto era así en lo que hoy en día se conoce como *Teorema de Bohm y Jacopini*, pero no rompieron lanza alguna para defender que programar en base al uso de las tres famosas estructuras fuera bueno y recomendable. Y todo el mundo siguió a su bola...

Hasta que uno de los científicos de la computación más reputados de entonces y de todos los tiempos, el holandés **Edsger W. Dijkstra**, publicó en 1968 su famoso artículo “**Go To Statement Considered Harmful**”, en el que cargó con armas y bagajes contra el uso y el abuso de la sentencia de bifurcación, abogando fuertemente por el uso exclusivo de las tres sentencias básicas.

La (reducida) comunidad informática de entonces aceptó bien, en general, la recomendación del maestro, pero seguía habiendo dificultades: todo esto está muy bien, sí, pero, ante una necesidad determinada, cuando hay que resolver un problema concreto, ¿cómo diseñamos y codificamos el programa usando exclusivamente las tres estructuras? En definitiva, *¿cuál debe ser el método a seguir?*

Por tanto, no fue hasta que se comenzaron a formalizar métodos de Diseño Estructurado de Programas que se comenzó a considerar en serio la posibilidad de utilizar la técnica en el mundo real.

El primero que realizó esta necesaria formalización fue el ingeniero francés **Jean Dominique Warnier**, a la sazón trabajando en Bull Informatique, que publicó en 1972 su libro “**Logique de Construction de Programmes**”.

Proporcionaba por primera vez un método sencillo y potente para diseñar un

[illegible]

Se fijaba en los datos que tenía que tratar el programa para así obtener la estructura correcta de tratamiento. Y obligaba siempre a especificar que cada bloque lógico tuviera un principio y un fin, aunque no hubiera nada que hacer allí y estuvieran vacíos. Esto último, por muy completo que fuese, fue lo que menos gustó al informático *de la calle*, como podéis suponer. Indudablemente, representó un gran avance.

En España, Bull tradujo del francés y publicó el libro de Warnier, de nombre “**La Programación Lógica**”, libro que yo compré, estudié, que cometí el error de prestar... y que no he vuelto a ver. En fin.

La publicación de este método no tuvo, sin embargo, la repercusión que hubiera debido por dos causas principales: primera, **no se centraba en ningún lenguaje concreto**, es decir, no daba instrucciones concretas sobre cómo llevar a la práctica sus predicados en Fortran, o Cobol, Assembler o lo que fuera. Y segunda, **¡era una cosa “not invented here”!** Como era obra de un francés y publicada originalmente en francés, el mundo anglosajón no entró mucho al trapo... Entonces, igual que ahora, la informática se escribía en inglés, en inglés *americano*, para más señas. Y Jean Dominique Warnier era francés,

había cometido la tropelía de publicar originalmente en francés... y para colmo, trabajaba en Bull, una empresa de la competencia... y francesa.

No fue hasta que **Ken Orr**, a principios de los ochenta, publicó su método, muy similar al de (o basado en el de) Warnier, denominado *Warnier-Orr* en la literatura, que el mundo anglosajón descubrió a Warnier... ¡y cómo!, puesto que fue adoptado por una de las Metodologías importantes de Desarrollo de mediados y finales de la década de los 80: *Method/I*, de Arthur Andersen (que aún no se había convertido en Andersen Consulting, ni mucho menos en Accenture).

*Me encantaba Method/I*: Como cuarenta o cincuenta o más, no me acuerdo, volúmenes de ochocientas o mil páginas cada uno... que nadie leyó nunca, claro, más allá de algún resumen. Pero especificaba todo, todo, todo lo que había que hacer, de qué manera documentar, quién debía firmar, etc., para asegurar que... para asegurar que... eeh...

...Bueno, en teoría para asegurar que *el proyecto se ejecutaba conforme a especificaciones, tiempo y coste* (cosa que nunca pasaba), así que en realidad lo que aseguraba era que... *en caso de que el proyecto se fuera al garete, tú, pobre director del proyecto, tuvieras la espalda cubierta*, mayormente. Pero esa es otra historia y será contada en otro momento.

Gracias a Orr se arregló la segunda de las causas antes citadas (el idioma y el “*not invented here*”); en cuanto a la primera de ellas, la falta de normas específicas de codificación, la solventó en 1974 **Marie Thérèse Bertini**, cuando publicó, junto a **Yves Tabourier** como coautor, “**Le Cobol Structuré**” (también en francés, claro está). Apenas se encuentra nada en la red para refrescarme la memoria, así que recurriré exclusivamente a ella.

Básicamente el “**método Bertini**” derivaba del de Warnier, ambos de origen francés y creo recordar que colegas en Bull durante una temporada: una vez conseguido el diagrama todo lleno de llaves, Bertini lo que preconizaba era, *aproximadamente*, reescribir la estructura del programa, pero ahora con “cerezas”, es decir, circulitos, para cada bloque lógico, conectados con líneas, pero ahora la jerarquía no era de izquierda a derecha, como ocurría con la técnica de Warnier, sino de arriba abajo, lo que resultaba más natural.

Sin embargo, supongo que para mantener algún tipo de compatibilidad con las llavecitas, inexplicablemente se empeñó en que los niveles se codificaran *de derecha a izquierda* y no de la forma natural para los occidentales, de izquierda a derecha. Igual Marie Thérèse tenía algún ascendiente judío o musulmán... Aunque, si he de ser sincero, siempre que he visto un diagrama de Bertini estaba escrito de izquierda a derecha, dijera el libro lo que dijera...

Pero lo importante de su método fueron sus normas para codificar la cosa resultante en Cobol... y sólo en Cobol; no publicó nada, que yo sepa, para ningún otro lenguaje. No eran muy complicadas, y básicamente se centraban en tres aspectos:

Por un lado, la **obligatoriedad de codificar los inicios y finales de cada bloque** (mediante sentencias **PERFORM INICIO-TAL-COSA** y **PERFORM FIN-TAL-COSA**, respectivamente), incluso aunque alguno estuviera vacío porque no hubiera nada que hacer allí, en cuyo caso llevarían sólo la instrucción **EXIT**, que no es tal instrucción, como imaginaréis, sólo sirve para que un párrafo vacío no esté... eso, vacío.

Por otro, la **prohibición de codificar PERFORM ... THRU ...** Esto forzaba a que para codificar una alternativa, hubiera que hacerlo así: **IF tal y tal PERFORM PARRAFO-DEL-SI ELSE PERFORM PARRAFO-DEL-NO**. Como consecuencia

generaba una gran profusión de procesos anidados en cascada, en ocasiones muy difíciles de seguir.

Y, por fin, **la prohibición total y absoluta para siempre jamás de usar el GOTO.**

La verdad es que yo nunca fui *bertiniano* practicante, aunque sí que me leí en su día lo que cayó en mis manos... ventajas de haber aprobado cuatro años de francés en el Instituto. Eso quiere decir que, treinta y muchos años después, no estoy seguro al 100% de si las tres reglas eran exactamente así, o me falta o sobra alguna. ¡Qué se le va a hacer! Habrá que conformarse con mi vieja memoria...

El método tuvo un cierto éxito y se enseñó bastante en centros de formación durante unos años. Pero no era perfecto. Bien está que se prohíban los GOTOs indiscriminados, pero es que, si quieres usar toda la potencia del Cobol, *en ciertos casos es obligatorio usar GOTOs*. Por ejemplo, si codificas con más de una sección, lo que es bueno y recomendable por claridad, necesitas en todas ellas, menos la principal, dirigirte al final físico de la sección para poder devolver el control al proceso llamador. Y no se me ocurre ninguna otra forma de hacerlo que con un GOTO. Es decir, usar Bertini implicaba *codificar todo el programa en una sola sección*, y cuando tienes dos o tres mil líneas de código sólo en la PROCEDURE DIVISION... eso no es exactamente una buena idea.

Además, existían ya en la época ciertos problemas que no habían recibido atención por parte de ninguno de aquellos incipientes metodólogos.

Hasta que, en 1975, el científico de la computación inglés **Michael A. Jackson** (que, que yo sepa, además de ser blanco durante toda su vida y de seguir perfectamente vivo en el momento de escribir estas líneas, *canta y baila fatal*), publicó su famoso libro **“Principles of Program Design”**. Y el tema se solucionó. Literalmente. Definitivamente.

Ya no se volvió a publicar ningún otro método general de Diseño Estructurado de Programas, pues no había mucho más que añadir al Método de Jackson. Resolvió tanto el procedimiento formal de diseño, dotándole de una lógica aplastante, como esos *“casos especiales”* que molestaban a los otros métodos de diseño estructurado. Y además, preconizaba una forma de codificar en los lenguajes más importantes de la época, significativamente Cobol, Fortran y PL/1.

Y después de todo este rollo, ¿no me quedo con las ganas de citar brevemente **cómo es el Método de Jackson!**, que por algo es mi preferido, además del más completo.

Lo más impactante del método es que *todo el diseño de la estructura del programa se basa en lo único que es completamente conocido, fijo y seguro en el momento de su escritura: los datos*. Efectivamente, todo programa que se precie y que sirva para algo debe leer y escribir ciertos datos, que estarán en cualquier tipo de soporte y que tendrán una estructura determinada, conocida y fija, pues, si no es así, malamente vamos a poder programar nada. A partir de la estructura intrínseca de los datos va siguiendo unos pasos concretos y muy bien definidos que terminan con el programa codificado en pseudocódigo, o directamente en el lenguaje que sea.

La idea es que, **dado un problema determinado que se aplica sobre ciertos datos conocidos, sólo haya una solución posible**. Como supongo suponéis, esto no es *exactamente así*... pero hay que reconocer que se parece mucho.

La notación que utiliza Jackson para representar las tres estructuras famosas en JSP es arborescente y bastante intuitiva:



Cada bloque individual puede estar a su vez descompuesto en cualquier otro tipo de estructura, lo que permite crear árboles tan grandes como sea preciso. Y, además, utilizando esta notación se pueden representar tanto datos como programas.

Veamos sucintamente su método, que preconiza cinco pasos secuenciales. En la página de la Wikipedia inglesa dedicada al JSP, "*Jackson Structured Programming*", hay un ejemplo de los diagramas de cada uno de los pasos que permiten ilustrarlo con algún ejemplo.

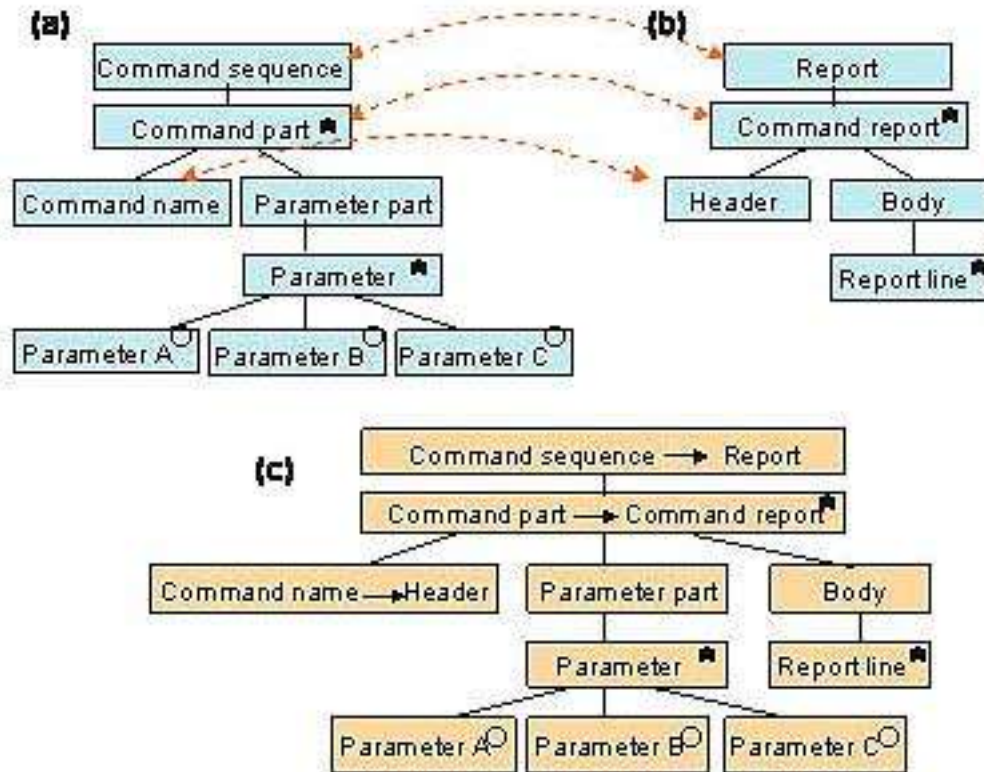
**1 Establecer la Estructura de los Datos.** De todos los datos involucrados, por separado, tanto los de entrada como los de salida.

**2 Establecer las Correspondencias entre Datos.** Para que haya correspondencia entre dos entidades de dos conjuntos de datos, sean de entrada o de salida, debe darse que: **a)** Ambas tengan el mismo número de elementos, y **b)** Estos estén en el mismo orden en ambas entidades, o sea, ordenados de la misma forma. *Hasta aquí, fácil, ¿no?*

**3 Crear la Estructura del Programa.** Para ello, cada entidad con correspondencias da origen a una única entidad, del mismo tipo, en la estructura final del programa, y el resto se insertarán en su lugar natural correspondiente, sin cambiarles de naturaleza, es decir, una secuencia en los datos dará origen a una secuencia en la estructura del programa, etc.

Como en este paso tratamos ya con procesos y no con datos, el propio nombre de las entidades cambia. La entidad de datos llamada "CUENTA", por ejemplo, ahora pasará a denominarse "PROCESAR CUENTA".

En la ilustración tenéis un pequeño ejemplo de cómo estamos hasta este paso.



*Tampoco esto es muy difícil...*

**4 Listar y asignar las operaciones individuales.** Jackson propone un checklist de operaciones, que debe cumplimentarse en orden, para que no se olvide nada:

- a) Instrucciones de finalización del programa.
- b) Instrucciones de cierre de ficheros.
- c) Instrucciones de apertura de ficheros.
- d) Instrucciones de escritura en ficheros.
- e) Instrucciones de lectura de Ficheros.
- f) Instrucciones de movimiento de campos, entre ellos los de las claves de ruptura.
- g) Instrucciones de cálculo.

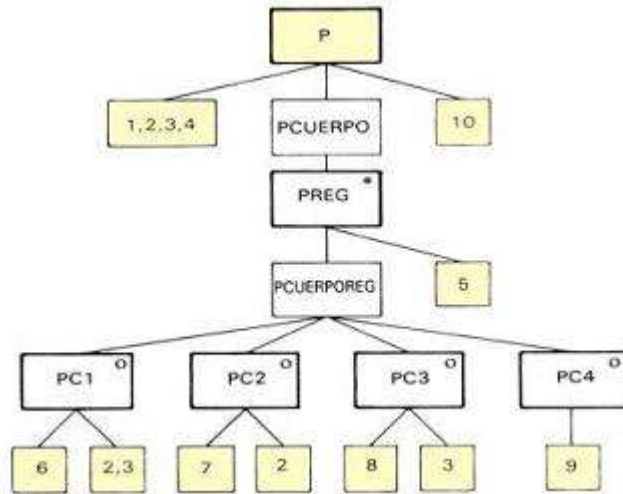
Una vez obtenida la lista, se asignan por riguroso orden las instrucciones a la estructura. *¿Dónde?* Siempre en secuencia y en aquel lugar donde la instrucción se ejecute una y sólo una vez, teniendo en cuenta las operaciones ya asignadas, para situarla en su orden lógico, antes o después de ellas. Por ejemplo, la lectura de un fichero debe ubicarse donde sea, pero siempre después de su apertura... Es fácil determinar el lugar exacto donde asignar cada instrucción si se asignan en el orden preconizado en el checklist.

*¿Cómo?* Con un círculo (otra *cereza*, vaya), dentro del cuál se inserta el número o código de la instrucción en la lista.

*Sencillo ¿no?*

El Resultado sería algo como el gráfico de la ilustración siguiente (bueno, normalmente sería *bastante más complicado*), en el que el autor ha sustituido los circulitos para representar las instrucciones, las *cerezas*, por cuadrados, por razones tipográficas,

supongo.



**5 Generar el pseudocódigo.** Muy *fashion* en esa época, e imprescindible para poder catalogar el método como “serio”, pero nadie escribía ya entonces el pseudocódigo, sino directamente el programa. Tiempo que se ahorra uno... Para ello, Jackson propuso un sistema de codificación de sus estructuras en los lenguajes más usados del momento, sobre todo Cobol, Fortran y PL/1.

¡Hala! **Ya tenemos el programa escrito.** Un auténtico chollo. Sencillo, rápido, bien definido... Además, catorce programadores diferentes a cuál más raro, obtendrán la misma estructura, puesto que los datos son como son, están perfectamente definidos, y todo lo demás se deriva de ellos... *Más o menos... O así...*

Además, Jackson, en su libro de 1975, resolvió una serie de problemas que se plantean habitualmente en computación. Entre ellos, preconizó la lectura adelantada (eso, ahora obvio, de leer el primer registro de cada fichero nada más abrirlo, lo formalizó él), incluso la doble o triple lectura adelantada, cuando sean precisas; y sistematizó el procedimiento para mezclar dos o más ficheros por clave (¡otra vez el *Padre-Hijo* a escena!). Pero además, formalizó la solución a otra serie de problemas habituales:

**a) El backtracking.** En esta estructura especial se trata un conjunto de datos *como si lo que hiciéramos fuera a valer*, pero *permitiendo cambiar de opinión cuando detectamos que nos hemos equivocado*. Puede parecer algo poco útil, pero es que exactamente eso es lo que ocurre en las transacciones online: hay que validar, en primer lugar, que todos los campos de entrada son correctos y que la acción demandada es factible; sólo cuando se ha verificado que todo es correcto se puede proceder a realizar las actualizaciones pertinentes.

La validación puede resultar bastante complicada, accediendo a diferentes Bases de Datos, etc. Además, en cuanto se detecta un error, se informa del mismo al peticionario y se termina el proceso de la transacción: no tiene sentido seguir validando y gastando ciclos de CPU cuando ya sabes que la transacción es errónea.

Diseñar este caso sin backtracking es... bueno, *es un tostonazo*, con ramas de alternativas bajo ramas de alternativas bajo otras ramas de alternativas... Sí, un tostón, y más aún cuantas más verificaciones sea preciso realizar, teniendo en cuenta que en una



transacción compleja pueden ser necesarias muchas verificaciones, dificultando, de rebote, su mantenimiento posterior...

Sin embargo, usar en su lugar un backtracking resuelve el problema de forma elegante y sencilla, como podemos observar en el sencillo ejemplo siguiente.

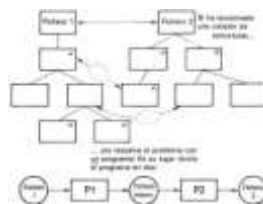


**b) La Programación Invertida.** (¡Y no valen chistes, que *bastantes* hacíamos ya nosotros entonces...!), que resuelve el problema de la colisión de estructuras, es decir, lo que ocurre cuando, en el paso 2 del método, no es posible encontrar correspondencias entre una entidad de la estructura de datos de entrada y otra entidad en la de salida.

La solución obvia para resolver estos problemas es, desde luego, **dividir el programa en dos** mucho más sencillos, usando para conectarlos un fichero intermedio.

Pero este fichero intermedio rara vez queremos (o podemos permitir) que exista físicamente. *Debe ser un fichero virtual*. Lo que sistematiza el método de Jackson es un procedimiento para eliminar físicamente ese fichero intermedio, pero que sigue existiendo lógicamente, es decir, **sin cambiar en nada ni la estructura ni la codificación de ambos programas**.

Este milagro se consigue “*invirtiendo*” uno de los dos programas respecto del otro, es decir, convirtiendo uno de los dos programas originales en una rutina de entrada (o de salida) del otro y comunicándose exclusivamente entre sí mediante el área del registro del fichero intermedio, que comparten ambos programas. En la imagen siguiente tenemos una ilustración del proceso de inversión.



Supongamos que hemos decidido convertir al primer programa “P1” en rutina invertida de entrada del segundo, “P2”, que quedaría en este caso como el principal. En este último P2 poco hay que hacer: cada vez que en él se lee el fichero intermedio se sustituye por una simple llamada a P1 (y se eliminan las *Open* y *Close*, obviamente). Únicamente hay que tener en cuenta cómo comunicar el fin del fichero: la solución obvia es pasar el registro con una marca (en Cobol, típicamente *HIGH-VALUES*, es decir, “FF’s” hexadecimales).

Pero, en el otro lado, para el programa P1, el programa P2, que ahora es el principal, es su propia rutina de salida... Así que, cada vez que en este programa P1 se debe escribir el registro del fichero intermedio, lo que ahora debe hacerse es *devolver el control a su llamador* (P2) que se encargará de “escribir” el registro y volverá a llamar a P1 cuando necesite un nuevo registro... e informándole, de paso, de que su registro ha sido escrito y que puede seguir su proceso por donde iba.

En una palabra, ambos programas se ejecutan simultáneamente, como estos dos niños jugando en un columpio, cediéndose mutuamente el control de la ejecución entre sí y comunicándose exclusivamente mediante el área del registro del fichero intermedio.

Es una solución de una elegancia suprema y una potencia inigualable. Además, contra lo que podría parecer, su uso no está restringido exclusivamente a los procesos batch. Puede utilizarse también (yo lo he hecho en bastantes ocasiones) en la resolución de transacciones online.

Y sin embargo, *siempre ha sido muy poco utilizada*, no por la dificultad de entender el mecanismo, que es realmente sencillo una vez se explica, sino por la aparente dificultad de codificar la inversión en el programa invertido, pues supone usar GOTOs *extraños*, incluso GOTOs condicionales (*GO TO DEPENDING ON*) que casi nadie quiere utilizar, porque dan *yuyu...* y se pierden **una de las técnicas más poderosas de programación jamás inventada**.

Bien, terminado el paseo por el **Método de Jackson de Estructuración de Programas**, desde entonces prácticamente no se ha avanzado más en la teoría, porque **todo había quedado establecido**. Jackson hizo un trabajo formidable y realmente no dejó mucho más que rascar.

Aquí: [http://www.ferg.org/papers/jackson--jsp\\_in\\_perspective.pdf](http://www.ferg.org/papers/jackson--jsp_in_perspective.pdf) podéis encontrar un interesante ensayo del propio Michael Jackson sobre “JSP en perspectiva”, escrito en el año 2001, que da un repaso al método a lo largo del tiempo.

Fue entonces, a principios de los ochenta, cuando nos pusimos en España a utilizar técnicas de Programación Estructurada en empresas e instituciones. Pero, antes de eso, hubo que dar a conocer la nueva técnica a los responsables de Informática, para convencerlos de que era mejor para sus negocios adoptarla.

Como podéis imaginar, en aquellos años los responsables de Proceso de Datos no estaban demasiado al tanto de las novedades que se producían en la profesión, primero, por el *dichoso inglés*, y luego, por su origen, pues casi ninguno de ellos era informático de profesión, sino que provenían de puestos administrativos y su formación era en otras áreas... quien la tenía, porque, por ejemplo, uno de los Directores de Informática que yo conocí comenzó su carrera en el Banco como botones, lo que dice mucho en favor de su capacidad.

Apenas se hacían salidas a Conferencias al Extranjero y no se leían muchas revistas especializadas americanas o inglesas, que además de ser incomprensibles (por el idioma, claro) eran caras... pero caras. Por lo tanto, el trabajo de promoción le correspondió a las Consultoras y/o Empresas de Formación, que, viendo en esta nueva técnica una vía de incrementar su negocio, promocionaron intensamente su propagación.

Ninguna entidad, que yo sepa, se adhirió al asunto de inmediato, pero ciertamente muchas comenzaron a hacer sus pinitos, sus pruebas para ver si la cosa de verdad funcionaba y mejoraba la productividad de la programación o era un canto más de sirenas, de los que de tanto en cuando aparecen aquí y allá. Entonces enviaron a recibir formación en las nuevas técnicas a uno o dos, quizá cuatro programadores “*escogidos*”, escogidos por las razones que fueran, en cursos abiertos que duraban tres o cuatro días. Resulta que yo fui uno de esos elegidos, a finales de los setenta, en el primer Banco en el que estuve.

Estos pocos programadores, al volver a su instalación, comenzaban a utilizar la nueva técnica, sobre todo por curiosidad, inicialmente de forma balbuceante, pero poco a

poco con mayor convicción. Hablo por mí y también por otros compañeros que trabajaban en otras instalaciones, pues la *Programación Estructurada* era un tema caliente de las conversaciones entre colegas de la época. Hicimos nuestros pequeños pinitos, sí, pero no conseguíamos convencer a nuestros mayores de que fuera conveniente la adopción generalizada de la técnica, entre otras cosas, porque éramos simples programadores... o sea, unos *mindundis*.

Y de pronto, algo ocurría: ante la necesidad que acontecía de vez en cuando de escribir un programa complicadísimo de resolver, de esos a los que nadie se atrevía a hincarle el diente, entonces te lo encargaban para que lo escribieras de forma estructurada, con la coletilla de “Hala (*listillo*: esto lo pensaban pero no lo decían), a ver si eres capaz de resolver este embolado con esa cosa rara nueva...”, y todo esto con *una aviesa sonrisilla de diablillo* que te llegaba hasta el hígado...

...Y entonces *el avisado programador*, tocado en su amor propio, se ponía a diseñar y codificar el maldito programa... y resulta que, sorprendentemente, en mucho menos tiempo del que se suponía el programa estaba listo, y encima funcionaba bien rápidamente (no, *a la primera*, no, que a la primera no funciona nunca nada...) y además no se eternizaba en máquina... Y entonces era cuando las *fuerzas vivas* comenzaban a darse cuenta de la potencia del asunto y a tomárselo en serio. Y ese primer adelantado contribuyó de forma importante a que la técnica se fuera imponiendo paulatinamente.

Por lo que yo sé, procesos similares fueron ocurriendo poco a poco en diferentes Centros de Cálculo, conforme las historias de éxito iban siendo conocidas y, además, aunque las entidades (los Bancos, por ejemplo) fueran competencia unas de otras, entre los informáticos de entonces había una sana camaradería, por lo que unos íbamos contando los unos a los otros nuestras experiencias, ayudando a desechar lo que no había funcionado y a extender lo que sí.

Así que, a lo largo de la primera mitad de los ochenta, las empresas se fueron convenciendo gradualmente de la bondad de la Programación Estructurada y sobre 1983-84 ya todos los Centros de Proceso de Datos importantes habían adoptado la técnica.

Se prepararon Planes de Formación para reciclar a los profesionales más antiguos (en algunos casos se contrataron empresas de formación, en otros se utilizaron recursos internos, depende de lo tacaña que fuera la empresa), y como consecuencia todo el mundo fue convertido a la nueva *religión estructurada*... aunque no todas las conversiones fueron completas, pues seguía habiendo adoradores de los antiguos dioses del GOTO que, siempre que podían, elevaban una ofrenda a su dios ancestral...

Muchos adoptaron el Método de Bertini, quizá no en su forma completa, pero sí al menos su principal mandamiento: “*El que ponga un GOTO, ¡al infierno de cabeza!*”. Una norma sencilla de entender por todo el mundo. No es ni mucho menos tan completa y eficaz como lo es el Método de Jackson, pero obligaba necesariamente a codificar de forma estructurada, así que muchas consultoras y empresas de Servicios adoptaron el Método, pues ahorra costes de formación a su personal y les permitía ser productivos lo antes posible.

Otras empresas (aunque no tantas) adoptaron el Método de Jackson; la posibilidad de usar backtracking, sobre todo en un entorno transaccional, facilitaba enormemente la codificación de las validaciones, y la programación invertida resolvía algunos problemas muy molestos. Vale, Jackson permite usar GOTOs, no cualquier GOTO, desde luego, sino en ciertos sitios y lugares y siempre “hacia abajo” nunca “hacia arriba”, según se explica en su libro original, el mítico “**Principles of Program Design**”. Y usar GOTOs no gustaba a

algunos, muchos de ellos, *desertores del spaghetti*, pero en fin.

La consecuencia fue que los buenos resultados que se obtuvieron eliminaron casi todas las reticencias en no mucho tiempo. Se programaba mejor, más rápido y los programas resultantes eran de mayor calidad, así que pocas dudas había...

En el mundo Jackson, todo el mundo hacía el “*mono*”, o sea, la estructura del programa con las instrucciones asignadas, a partir del cuál se codificaba. Aparecieron diversos precompiladores y programas, entre ellos JSP-COBOL del propio Jackson, que ayudaban al diseño y codificación de programas escritos según este método.

Pero ninguno tuvo mucho éxito entonces, al menos en España, por una razón evidente: **¡No había PC's!** Y los pocos que había, carísimos y con muy poca potencia, se dedicaban a otros menesteres, no para que un vulgar programador pintara su “*mono*”.

Una pena. Si hubiéramos dispuesto entonces de las facilidades que hubo sólo cinco, seis o siete años después, cuando se generalizó el uso de PC's en Proceso de Datos, seguramente otro gallo hubiera cantado. Pero las cosas fueron así, porque ahora, cuando sí es posible fácilmente preparar un programa para, visualmente, diseñar y codificar el programa estructurado... **no hay interés**.

Por ejemplo, Henrik Engström, de la Universidad sueca de Skövde, ha puesto a la disposición de quien quiera usarlo un programita *freeware* que **permite diseñar programas usando el Método de Jackson** y generar luego el código en C o Pascal.

Es posible encontrarlo (o al menos lo era) en la dirección de internet: <http://www.his.se/english/university/contact/staff/henrik-engstrom/jsp-editor/>. Pero pocos lo encontrarán hoy de utilidad más allá de cómo una anticuada curiosidad.

Entonces, a falta de precompiladores o productos que ayudaran en el diseño y codificación, los “monos” (los de Bertini o los de Jackson) se hacían a mano y los programas se codificaban conforme a las prescripciones del método correspondiente... y a las manías y querencias del programador de turno. Por eso las empresas y las instituciones de la Administración empezaron a emitir **normas de programación**, en las que se especificaba qué método de diseño se debía utilizar y, lo más importante, cómo codificar el programa resultante.

Tanto Bertini como Jackson proponían un método de codificación en Cobol, el lenguaje más usado con mucho en la época, así que en muchos casos se adaptó la normativa a lo que decían ambos métodos, aunque ciertamente en otros se usaron normas diferentes, o una sana mezcla entre ambas posibilidades.

Además de prohibir los GOTOs o restringirlos a los usos especificados en el método de Jackson, y especificar cómo codificar Secuencias, Alternativas y Repeticiones (y Backtrackings y tal, en su caso), se dieron normas de nomenclatura, sobre cómo llamar a los procesos y campos de los programas, así como a los ficheros, impresos, bases de datos y a los propios programas en sí.

Todo este esfuerzo normativo fue muy importante: en unos momentos de enorme actividad, pues muchas empresas estaban construyendo Sistemas de Información nuevos, de los que una parte considerable permanecen activos hoy en día, **asegurar una serie de criterios unificados de nomenclatura y codificación redundó en una mucha mayor facilidad de mantenimiento posterior**.

En el capítulo dedicado al Cobol cité el hecho de que programas escritos hace veinte o veinticinco años sigan funcionando, tras años y años de modificaciones, y que sigan siendo mantenibles. El uso de toda esta batería de normas rígidas a lo largo del tiempo ha

facilitado, si no directamente *permitido*, que esto sea posible.

Muchas empresas no sólo se quedaron en la publicación de las normas y la obligación más o menos “*moral*” de su uso: escribieron programas que verificaban que los programas que se ponían en Producción cumplieran escrupulosamente la normativa, y si no... en general impiden su puesta en Producción; pero otras veces se limitan a informar al autor para que se arregle el desaguisado, pero se permite su uso, de momento. Esto es así para facilitar el arreglo de un *programa malvado* a las tres de la mañana...

Tomar una u otra opción depende de la instalación, ambas son en principio válidas. Pero lo que sí es cierto es que, una vez que un programa está en Producción y funciona como es debido, nadie lo va a tocar *motu proprio* sólo para arreglar su adecuación a las normas, con la cantidad de trabajo que hay que hacer... Por eso lo generalmente preferido es que el *Portero del Cotarro* no permita el uso en Producción del programa hasta que esté convenientemente arreglado y quede de acuerdo a toda la normativa de codificación y nomenclatura de la empresa... sea la hora que sea.

...Y claro, dada una norma, siempre hay quien se rompe la cabeza... para romperla. Parece que una de las características del carácter de los españoles es saltarse, por puro placer, las normas establecidas, sólo por el hecho de hacerlo (ignoro si por ahí fuera esto ocurre también, aunque sospecho que mucho menos que acá). Y fuimos capaces de saltarnos las normas, ya lo creo. Por puro deporte, además, sin ningún ánimo de ser perniciosos.

Cuando no había programas que comprobaban el cumplimiento de las normas, era muy fácil: bastaba con inventarse un caso donde con un par de GOTOs se evitaba una estructura compleja... así que ya teníamos nuestra *coartada*. Pero cuando se implantaron los programas de control, ya no era tan fácil... y sin embargo lo conseguíamos.

Por ejemplo, para meter un GOTO y que se lo tragara el *Portero*, poníamos la G en la columna 72 de una línea, y en la siguiente línea poníamos un guión en la columna 7 (en Cobol esto significa que es una línea de continuación y debe añadir su contenido al de la línea anterior: se usa sobre todo para literales largos) y después la O en la columna 12. Raro, ¿no?

El compilador recomponía el GO de ambas líneas y lo interpretaba sin problemas... ¡pero el programador del programa de verificación no había pensado que un verbo GO no estuviera en una sola línea! (Normal, *¿a quién se le ocurre, si no es para fastidiar...?*). Y así engañábamos al sistema... que en definitiva era engañar a nuestros propios compañeros, o sea, *a nosotros mismos*... pero así nos lo pasábamos bien. En descargo mío, debo confesar que, las pocas veces que lo hice, lo arreglé en la siguiente modificación... Hubo quien no.

Y así, años más tarde, somos quienes somos, estamos donde estamos y hacemos lo que hacemos... porque *venimos de donde venimos*.

**¿Se usa actualmente la Programación Estructurada?** Pues sí. Más o menos. Nadie pone GOTOs, que definitivamente son considerados por todo el mundo “*harmful*”, como Dijkstra decía en su día, cuarenta años ha. Pero se trata de una programación estructurada “intuitiva”, poco apoyada en fundamentos teóricos... porque no se enseñan, que yo sepa.

**¿Se usan los métodos de diseño estructurado de programas?** Pues no creo, al menos tal como fueron definidos. Hace años que no veo a ningún programador pintar un “mono” de ningún tipo. Las pocas veces que *Yo* hago uno, me miran como si viniera de

Marte... Quizá porque ahora son mucho más inteligentes de lo que nosotros éramos y no lo necesitan. Porque nosotros... nosotros **sí** que lo necesitábamos.

¿**Es conveniente que se enseñen estos métodos?** Bajo mi modestísimo punto de vista, desde luego que sí. Es más, cuando alguien comienza a estudiar informática, **ESO sería lo primero que yo le enseñaría y se lo grabaría a fuego para que no se le olvide nunca**. Porque sigue siendo útil; más que útil, **necesario**. Por mucho que el paradigma de Orientación a Objetos haya cambiado la forma de concebir y diseñar las Aplicaciones, cada proceso que se debe realizar, lo queramos o no, sigue siendo un proceso susceptible de ser estructurado. Sobre todo si es un proceso complejo, que debe acceder a Bases de Datos, validar una serie de campos, generar una respuesta...

Y del batch, ya, ni hablamos. Porque resulta que *sigue haciendo falta ejecutar procesos batch*. Y ahí sí que no hay excusa. Y se ve cada cosa por ahí codificada... yo me he encontrado con barbaridades como para erizarte el cabello de la peluca.

Sin ir más lejos, y esto es un *caso real*, acceder a la Base de Datos de Cuentas *una vez por cada movimiento* de la Cuenta, en lugar de *una vez por cada Cuenta*. Y eso es, ni más ni menos, por no asignar correctamente la instrucción de lectura (la `SELECT cuenta... FROM TABLA-CUENTAS WHERE ...`) en su lugar correspondiente de la estructura. Y eso es, a su vez, por **no saber**. Ni más ni menos. Y el programa funcionaba perfectamente, claro que sí. Utilizando ocho o diez veces más de los recursos necesarios, pero funcionar, funcionaba.

Ahora bien, lo peor de todo es que, cuando avisas al programador (*amablemente*, lo prometo) para que modifique el dichoso programa, porque está mal parido y consume muchos recursos, ¡encima se mosquea!, que esto... *esto* es lo que menos entiendo. En fin.

Y colorín, colorado, la Programación Estructurada se ha acabado... Lo próximo que viene no va mucho de *batallitas*. Éstas volverán más adelante, pero el próximo capítulo lo dedicaré a contaros qué es eso del “**Sort**”, que tantas veces aparece en mis notas y que tantos quebraderos de cabeza nos ha dado a los informáticos desde hace tantos años.

## 8 - El Sort (o el viejo problema de ordenar las cosas)

Ah, el *Sort*, ese viejo conocido... Voy a describir aquí qué es y para qué sirve el ubicuo *Sort*, que siempre aparece en las historias de informáticos, sean estos viejos, como el que suscribe, o jóvenes.

Si eres informático o estás en ello es muy posible que lo que viene a continuación te aburra, aunque quizá encuentres algo nuevo, quién sabe, porque voy a hacer un poquitín de historia... Y si no eres informático quizá te ayude a entender en alguna medida los mecanismos mentales que nos gastamos para enfrentarnos a uno de los problemas más incordiantes de la profesión: **la gran cantidad de tiempo que se “pierde”** (que *los ordenadores “pierden”*, en realidad, o mejor, “*invierten*”) **ordenando las cosas...**

A finales del Siglo pasado, una Consultora publicó un estudio que había realizado para averiguar en qué se consumía la CPU de los ordenadores de las empresas. Sospecho que, sobre todo, basó su estudio en clientes con mainframes de IBM, aunque sólo sea por lo sencillo que es en estos sistemas saber con exactitud, gracias a un producto llamado RMF, lo que consume cualquier programa a lo largo del tiempo.

A pocos sorprendió que **el programa que más recursos consumiera fuera el Sort** (es decir, el programa standard para ordenar los registros de un fichero, Base de Datos, etc), pero lo chocante fue hasta qué punto lo era: alrededor del 60% de todos los recursos de los ordenadores centrales de las compañías se destinaban a ordenar las cosas. Hablo de memoria, porque no he conseguido encontrar el documento ni vivo ni muerto, ni en papel ni tampoco en la red.

No debería, en realidad, resultarnos nada sorprendente: lo que los angloparlantes llaman “*Computer*” (artefacto para hacer cálculos), para franceses y españoles es un “*Ordenador*” (cacharro cuya función es ordenar cosas).

Por cierto, el término *computer* lo aplicaban los anglosajones para designar a las personas que “computaban” una serie de cálculos con una calculadora (*calculator*) o con una simple regla de cálculo. A partir de principios de los cincuenta del siglo pasado, los modernos sistemas fueron reemplazando paulatinamente a los *computers* humanos, que no sólo se quedaron sin trabajo, sino que también les canibalizaron el nombre...

En definitiva, el **Sort es el programa para ordenar ficheros** en todo Sistema Operativo que se precie. Y, que yo sepa, en todos ellos se llama de igual modo: **SORT**. Desde el B1 del NCR Century 200, hasta el MS/DOS o el UNIX, pasando por el MVS (o sea, z/OS).

Toma un fichero (o varios) de entrada, lo clasifica por los campos clave que se le indican, o por todo el contenido del registro si no se especifica nada, de forma ascendente o descendente... Y además, en algunos Sistemas como MVS, el propio programa sirve para hacer tratamientos a los ficheros: Unificar ficheros, Dividirlo en varios, Extraer registros... un programa muy completo, como veis. Y que se usa muchísimo.

Una precisión: los informáticos vejestorios (al menos *este informático vejestorio*) no decimos mucho “ordenar” en este contexto, sino más bien “clasificar”. Ya, ya sé que “clasificar” es distinto de “ordenar”, y que si traducimos literalmente al inglés “Algoritmo de Clasificación” nos sale “Clustering Algorithm”, muy usados actualmente para realizar

estudios de “Data Mining”, pero nunca “Sorting Algorithm”, que es la forma correcta de referirse a estos procesos en inglés.

Lo que definitivamente no se debe decir nunca, nunca es “*sortear*”. Os puede parecer raro, pero yo lo he escuchado muchas veces: “Hay que sortear este fichero”, como si hubiera que rifarlo o jugárselo a la carta más alta... Claro que también decimos mucho que “*displayamos*” un mensaje, “*seteamos*” una variable, o “*deleteamos*” un fichero...

Así que, como la costumbre es la costumbre, a partir de ahora, para no repetirme mucho, diré a veces clasificar y a veces ordenar.

En la vida real se clasifican cosas continuamente, bien para ordenarlas de cara a su comercialización, bien para separarlas por tamaños, tipos o cualquier otra característica.

Pues igual cuando usamos un ordenador: continuamente estamos clasificando cosas, o el Sistema Operativo lo hace por nosotros incluso cuando no nos damos cuenta. Cada vez que, en el Explorador de Ficheros de vuestro Sistema Operativo favorito, clicáis en las cabeceras para ver los ficheros ordenados por fecha, o por nombre o lo que sea, el Sistema invoca internamente al Sort.

Además, todas las Bases de Datos, para poder acceder rápidamente a los datos solicitados, requieren tener toda la información previamente ordenada o, al menos, tener ordenados los *índices de acceso a la información*, pero ésa es otra historia y será contada en otro momento. Cada vez que se requiere recuperar información de una Base de Datos o de un fichero para, por ejemplo, presentarla en pantalla, casi siempre se requiere clasificarla (casi seguramente mediante la conocida cláusula “ORDER BY”), porque *los humanos esperamos siempre encontrar la información en cierto orden*. Todo fichero que se envíe a algún lugar debe estar ordenado. Los procesos batch que toda empresa realiza (la liquidación de cuentas, la facturación, etc.) suelen requerir clasificar una y otra vez los ficheros intermedios. El Sistema Operativo necesita, para realizar sus funciones, mantener ordenado un montón de información...

En definitiva, para que los Sistemas informáticos funcionen como es debido se requiere que la información esté clasificada... siempre. Y ordenar registros de un fichero o de una Base de Datos es un proceso no muy complicado de entender, pero que **consume una enorme cantidad de recursos de máquina**.

Intentaré describir el problema que esto representa de la forma más sencilla que pueda y sepa, y también cómo los informáticos nos hemos ido buscando la vida a lo largo del tiempo para clasificar datos de forma cada vez más eficiente... a costa de hacer los algoritmos de ordenación cada vez más complicados, claro. Empecemos, pues.

Cuando no se habían inventado las Bases de Datos (por increíble que parezca, hubo un tiempo pretérito en que éstas no existían) y los ficheros maestros estaban en cinta magnética, para actualizarlos había que leer la última versión, digamos la  $n$ , en un armario de cinta, los cambios habidos en el día en otro armario de cinta, y grabar el fichero maestro resultante, la versión  $n+1$ , en otro armario más.

El proceso, del que ya hablé otras veces en capítulos anteriores, se conoce como una **actualización “padre-hijo”**, nombre que muy imaginativo no es, pero sí un rato descriptivo. Puede haber, además otros ficheros adicionales con otra clase de información, que deben ser leídos también coordinadamente con los ficheros principales para realizar correctamente el proceso.

Pues bien, es imperativo para que este proceso pueda funcionar que todos los



ficheros que intervienen en el proceso estén ordenados de la misma manera, por ejemplo, por número de cuenta. Si una cuenta no tiene ningún movimiento en el fichero de movimientos, se copia tal cual al nuevo fichero maestro. Pero si tiene movimientos, deben aplicarse a la información anterior que está en el “padre” (cambios de domicilio, de titulares, etc) y grabar la información resultante en el fichero “hijo”, incluso no grabar nada, en el caso de que el movimiento sea para “Dar de Baja”.

La única forma de saber si una cuenta tiene o no movimientos es ir leyendo en el mismo orden todos los ficheros de forma coordinada, en un proceso que, en sí mismo, se llama, obviamente, “**Merge**” (Fusión, en español, aunque nadie usa tal nombre).

Como los movimientos se recibían de las oficinas (y se grababan, por tanto) en cualquier orden, casi siempre según se iban produciendo durante el día en la oficina, era preciso en primer lugar clasificarlos para realizar el proceso de incorporarlos a los Ficheros Maestros. Y es completamente normal que hubiera que ordenarlos varias veces, por diferentes cosas cada vez, a lo largo del proceso. Por ejemplo, para liquidar los movimientos en cuenta corriente, estos deben estar clasificados por cuenta y fecha de valor; para contabilizarlos, por cuenta contable y fecha de contabilización; para los envíos de las comunicaciones a clientes, por el código postal, etc.

Por tanto, **clasificar registros fue ya una labor fundamental desde los primeros ordenadores**, y lo sigue siendo en la actualidad.

Es más, en los primeros pinitos del cálculo automático, y es obvio que me estoy refiriendo a la máquina tabuladora diseñada por Herman Hollerith para realizar el censo de Estados Unidos de 1890, considerada como el primer ordenador que merece tal nombre, para lo que servía realmente era para poder agrupar, según diferentes criterios, toda la información de todos los ciudadanos que se tenía almacenada en tarjetas perforadas: por nombre, sexo, raza, religión, condado, etc. En una palabra, clasificar la información por criterios diferentes cada vez. O sea, el primer ordenador era en realidad una clasificadora de tarjetas perforadas.



Aquí tenemos una imagen de la máquina tabuladora de Hollerith (Wikipedia, Jennifer, CC by SA 2.0).

Bueno, y, entonces... **¿cómo nos las apañamos los informáticos para clasificar datos?**

...Usando **Algoritmos de Clasificación**, naturalmente.

Aquí viene uno de los invariantes de la profesión: El problema es sencillo, está bien definido, es tan antiguo como la propia existencia de los ordenadores y, por analogía con cualquier otro área de la tecnología, parece que *debería haber consenso práctico sobre cuál es la mejor manera de hacerlo* y ser utilizada por todo el mundo.

Pues no. No hay un único algoritmo. Hay **muchos**, cada cual con sus ventajas e inconvenientes, más o menos difícil de programar, más o menos adecuado a cada tipo de lista a ordenar y más o menos eficiente. De todos ellos, y desde luego de todos los que cite yo en este capítulo, se encuentra muchísima documentación técnica en la Red, particularmente en la Wikipedia, tanto la española como la inglesa, aunque en general suele estar más completo en la inglesa, como de costumbre.

Describiré los algoritmos más importantes a continuación, pero, antes de eso, debemos contestar a una pregunta muy sencilla...

***¿Cómo ordenamos los humanos una lista de cosas, números, por ejemplo?***

Supongamos una lista de cifras que hay que ordenar de forma ascendente, es decir, deseamos colocar los números más bajos a la izquierda y los altos a la derecha (obviamente, la ordenación descendente es idéntica en cuanto al procedimiento, salvo que cambiamos “el mayor” por “el menor”).

Aquí tenemos un ejemplo de lista a ordenar que nos servirá durante todo el resto del capítulo.

Lista a ordenar

03	07	11	02	09	01	08	05	10	06	04
----	----	----	----	----	----	----	----	----	----	----

Veamos: Se trata de una lista pequeña, de sólo once elementos, así que seguramente la forma más sencilla de ordenarla es elegir de la lista cuál es el número más pequeño y copiarle en la primera posición a un nuevo “molde” del mismo tamaño.

En nuestro ejemplo, tomamos el “01” y lo copiamos el primero. Incluso podemos marcarle, para no equivocarnos y así evitar volver a tomarle de nuevo más adelante. Seguimos con el siguiente más pequeño (el 02), luego el siguiente... hasta que terminamos.

Lo que queda al final es, obviamente, lo siguiente:



Bien, pues ahora pensemos que esto mismo lo tiene que hacer un ordenador. Queremos escribir un algoritmo (un programa, una lista ordenada de instrucciones, en definitiva) que nos permita ordenar esta lista. En seguida resulta evidente que, para escribir tal *obra de arte*, hay que tener en cuenta ciertas restricciones que no teníamos en nuestro pequeño ejemplo:

1) Como el número de elementos a ordenar rara vez es fijo (dependerá de cuántas cuentas estamos tratando, que variará de un día a otro, de cuántos ficheros, cuántos números, etc.), nuestro algoritmo debe servir, sin cambios, para ordenar listas de cualquier tamaño. Desde listas de un solo elemento hasta listas tan grandes como queramos, pasando por listas de dos, de siete, de once... ¡o ninguno! Particularmente, hay que tener en cuenta que podemos vérnoslas con listas de bastantes millones de elementos, lo que introduce severas restricciones de espacio.

2) Como consecuencia, raras veces es factible pensar en duplicar la lista inicial en otra de igual tamaño: la memoria del ordenador es finita y cara, o al menos lo era. Por tanto, es una buena idea que nuestro algoritmo trabaje exclusivamente en la propia lista original, intercambiando elementos entre sí sin necesidad de espacio adicional. En realidad **sí** que necesitaremos espacio adicional, para un elemento solamente, que usaremos como memoria temporal para intercambiar dos elementos entre sí... salvo que los intercanbiemos con instrucciones del tipo “XOR”, pero esa es otra historia y será contada en otro momento.

3) Ya puestos, debemos tener en cuenta que podemos necesitar ordenar los elementos por algo que no sean números: Nombres, Direcciones, Códigos, etc. Y de cualquier longitud, aunque en cada ordenación habrá que avisar a nuestro algoritmo cuál es

el tamaño del elemento. Es más, es posible que nuestros elementos contengan varios campos diferentes asociados y sólo deseemos ordenarlos por uno de ellos. Por ejemplo, es el caso de la ordenación en el Explorador del Sistema: al ordenar la lista de ficheros por fecha, no sólo ordenamos las fechas, sino también toda la información del fichero asociada.

4) Además, podemos encontrarnos con varios elementos con la misma clave, que deberán quedar en la lista resultante todos juntos, y nuestro algoritmo debe estar preparado para ello.

5) Por fin, como podemos necesitar ordenar listas con gran cantidad de elementos, necesitamos que nuestro algoritmo sea *eficiente*, o sea, que haga su trabajo en la menor cantidad de tiempo posible. *Haciéndolo bien*, se supone.

El algoritmo más sencillo (y uno de los más antiguos) para ordenar una lista de elementos es el “**Algoritmo de la Burbuja**” (*Bubble Sort*).

Es muy sencillo e intuitivo: se basa en, comenzando por el primer elemento, ir sistemáticamente comparando cada elemento de la lista con el siguiente. Si están ordenados entre sí, se dejan como están; y si están desordenados, se intercambian uno con el otro. Después se pasa al siguiente elemento, y se repite lo mismo una y otra vez hasta llegar al final de la lista. O sea, tras la primera pasada, resulta esto:

Lista a ordenar										
03	07	11	02	09	01	08	05	10	06	04
03	07	02	09	01	08	05	10	06	04	11

Efectivamente, las dos primeras comparaciones (el 03 con el 07 y el 07 con el 11) han encontrado que el orden relativo de los elementos era correcto, así que no han cambiado nada. Pero en la tercera posición, en cambio, se encontraba el que a la postre ha resultado ser el mayor de todos los elementos (el 11), así que en las comparaciones sucesivas ha sido cambiado una y otra vez con los elementos siguientes, hasta llegar al final de la lista. Primero con el 02, luego el 09, y así hasta el 04 final.

Por tanto, en esta primera pasada el elemento de clave más alta (el 11) ha quedado correctamente situado en el final de la lista... pero los demás siguen, lamentablemente, desordenados. El nombre de “*burbuja*” viene precisamente porque el elemento de valor mayor de la lista se va desplazando a su través directamente hacia un lado como una burbuja se deslizaría por un tubo lleno de líquido hasta llegar a la superficie.

Es decir, en esta primera pasada hemos dejado correctamente ordenado un solo elemento: el de clave mayor de todos. Así que, para clasificar los elementos que quedan, no hay más remedio que **dar nuevas pasadas de ordenación** a la lista.

Veamos ahora lo que resulta tras las cinco primeras pasadas:

Original:	03	07	11	02	09	01	08	05	10	06	04
Pasada 1:	03	07	02	09	01	08	05	10	06	04	11
Pasada 2:	03	02	07	01	08	05	09	06	04	10	11
Pasada 3:	02	03	01	07	05	08	06	04	09	10	11
Pasada 4:	02	01	03	05	07	06	04	08	09	10	11
Pasada 5:	01	02	03	05	06	04	07	08	09	10	11

Efectivamente, queda claro que **en cada pasada sucesiva se ha conseguido ordenar efectivamente un solo elemento**, el mayor de los que quedaban desordenados (los sombreados), por lo que es fácil deducir que para ordenar completamente la lista necesitaremos dar tantas pasadas como elementos tiene, menos una (la última pasada no hay que darla, pues sólo queda un elemento por ordenar... y ya está, obviamente, ordenado), en nuestro caso, 10 pasadas.

Como en cada una de ellas examinamos los 11 elementos, este sencillo algoritmo necesita examinar ( $10 \times 11 = 110$ ) elementos para conseguir ordenar la lista. En términos técnicos, su *complejidad* es del orden de  $O(n^2)$ . Si hablamos de listas de unas decenas de elementos esto no es ningún problema, pero, si necesitamos ordenar centenares de miles de elementos... o millones...

Desde muy temprano se introdujeron mejoras obvias a la provecta burbuja para hacerla más eficiente. Entre ellas:

- No tiene sentido comparar siempre, a partir de la segunda pasada, todos los elementos, puesto que sabemos positivamente que el último ya está correctamente situado. Así que, en cada pasada, **reducimos en uno el número de elementos a revisar**. Con esta sencilla medida reducimos a la mitad el número de elementos revisados y se reduce casi en la misma proporción el tiempo necesario para ejecutar el algoritmo.
- Podemos, en cada pasada, **comprobar simplemente si hemos cambiado algún elemento con otro**, mediante, por ejemplo, un indicador o “*switch*” que inicializamos a cero al principio de cada pasada y que cambiamos a uno cada vez que intercambiamos dos elementos. Si después de una pasada completa el indicador sigue a cero, eso significa que la tabla ya está clasificada, pues no hemos tenido que cambiar ningún elemento de sitio y **podemos ahorrarnos el resto**.

Para comprobarlo, volvamos a nuestro ejemplo: tras dos pasadas más, es decir, siete en total, tenemos:



Original:	03	07	11	02	09	01	08	05	10	06	04
Pasada 5:	01	02	03	05	06	04	07	08	09	10	11
Pasada 6:	01	02	03	05	04	06	07	08	09	10	11
Pasada 7:	01	02	03	04	05	06	07	08	09	10	11

...o sea, ya tenemos la lista ordenada, aunque aún no lo sabemos; hasta que hagamos la pasada número 8 y constatemos que no hemos intercambiado ningún elemento, no sabremos que hemos terminado.

Es muy normal en las operaciones reales del mundo real clasificar una lista que *ya está clasificada* o casi clasificada, por lo que añadir este indicador es vital en todo buen algoritmo que se precie siempre que se pueda, pues hay algoritmos donde esta técnica no es posible.

- En cada pasada podemos, mientras realizamos el proceso general, averiguar también cuál es *el elemento de clave más pequeña*. Como hemos de revisar todos los elementos, podemos fácilmente anotar dónde se encuentra el más pequeño de todos y, al final de la pasada, **intercambiar el primer elemento de todos por ése más pequeño**. Entonces, en la segunda pasada se comenzará el proceso por el segundo elemento en vez del primero, en la tercera, por el tercero, etc.

Esto reduce el número de pasadas a la mitad, pues ahora se clasifican *dos* elementos por pasada, lo que mejora el tiempo ostensiblemente a costa de una mayor complejidad en el algoritmo, que en su parte interna debe ahora realizar dos comparaciones para cada elemento en vez de una, lo que también consume tiempo de máquina.

- Podemos **ir cambiando la dirección de la pasada** (arriba-abajo y luego abajo-arriba, obteniendo así un tipo de algoritmo llamado *Gnome Sort*) cada vez que realizamos un cambio, lo que mejora levemente el tiempo requerido por la burbuja.

- También podemos ir comparando no un elemento con el siguiente, sino con *uno que esté algo alejado de él* (por ejemplo, ir comparando el elemento  $n$  con el  $n+5$ , en lugar de con el  $n+1$ ). Esta sencilla modificación a la burbuja no fue “descubierta” hasta nada menos que 1991, y sorprendió a la profesión por sus excelentes resultados.

Con este sistema se intenta eliminar en lo posible las “tortugas”, que son aquellos elementos con clave pequeña que acabarán colocados al principio de la lista, pero que están inicialmente situados al final.

Por el propio funcionamiento de la burbuja, estas “tortugas” *sólo suben una posición hacia arriba en cada pasada*, lo que ralentiza mucho la terminación del algoritmo. Si os fijáis en nuestra lista de números veréis cómo el “04” va subiendo lentamente desde el final, pasito a pasito, hasta alcanzar su posición final en la tabla ordenada.

Este algoritmo (de nombre *CombSort*) mueve con rapidez las “tortugas” hacia arriba, por lo que tiene un excelente rendimiento, a cambio de una mayor complejidad de programación, comparado con la burbuja... aunque siempre compensa, desde luego.

Por fin, existen otros algoritmos que hacen más o menos la misma cosa de formas

diferentes y que, según el caso funcionan mejor que la burbuja, como son el *Ordenamiento por Inserción*, el *Ordenamiento por Selección*, etc., pero ninguno de ellos es muy eficiente, pues todos tienen una complejidad del orden de  $n^2$ , es decir, crece con el cuadrado del número de elementos de la lista.

...O sea, **mucho**. Cuando tratamos con grandes volúmenes de información, esta complejidad cuadrática implica un tiempo elevado para realizar la ordenación, lo que es casi siempre inaceptable.

Los informáticos rara vez nos conformamos con lo que tenemos, así que desde el principio nos pusimos a solucionar este problema. *Se pusieron*, en realidad, que uno es viejo, pero no tanto. Uno de los gurús de los albores de la Informática, el inglés C.A.R. Hoare (aunque nació en Colombo, en Ceilán, actual Sri Lanka), solucionó el problema nada menos que en 1960. Y digo *lo solucionó* porque no sólo describió el algoritmo, el ínclito Quicksort, sino que, algún tiempo después, *demostró matemáticamente que era el más rápido posible* para ordenar listas.

Voy a describir brevemente al Quicksort, un algoritmo realmente apasionante.

El **Quicksort** se basa en el famoso aforismo de Cayo Julio César: “**Divide y Vencerás**”. Y eso es exactamente lo que hace el algoritmo:

**1** Elegimos un elemento, en principio cualquier valor de la lista, cuyo valor usaremos como **pivote**. Unas implementaciones toman el primero, otras el último... a mí siempre me ha gustado más elegir el elemento que esté físicamente en la mitad de la lista a ordenar, ya veréis por qué.

**2** Recorremos la lista entera, con el objetivo de **colocar los elementos de clave menor que el pivote a la izquierda, y los mayores, a la derecha**. De hecho, en las mejores implementaciones, como la de Sedgwick, esta fase suele hacerse recorriendo simultáneamente la lista en las dos direcciones; cada vez que se encuentran un par de elementos descolocados, se intercambian entre sí.

**3** Acabado este proceso (ojo, que *no es exactamente tan sencillo como parece*), lo que queda es **un conjunto de elementos menores que el pivote**, luego **el pivote** (si hay claves duplicadas, todos los elementos de clave igual a la del pivote: todos estos elementos “pivote” ya está(n) correctamente colocado(s) en su posición final), y luego **otro conjunto de elementos con clave superior al pivote**. Conseguir esto es lo óptimo, pero, dependiendo de la implementación, puede quedar la lista dividida en sólo dos conjuntos: el primero, con los elementos de clave menor o igual que el pivote, y el segundo, con los elementos de clave mayor o igual. Es algo menos eficiente, pero también menos complejo de programar.

**4** *¿Y ahora?* Pues ahora hacemos lo que hacemos los informáticos desde el principio de los tiempos siempre que podemos: *reutilizamos el código*. Es decir, le aplicamos exactamente el mismo proceso descrito antes (de **1** a **3**), primero al primer conjunto de elementos (los de clave menor que el pivote), y luego al segundo (los de clave mayor que el pivote). Y ya está.

*Fácil, ¿no?*

Pues, para variar, **no**.

Esto de que *un algoritmo se llame a sí mismo* se llama **recursividad** y, aunque es una técnica muy potente, siempre nos da un poco de *yuyu* a los de la profesión, porque hay

que controlar muy, muy bien cuándo hay que terminar el proceso para evitar meter el programa en un hermoso bucle infinito, con lo que eso nos gusta...

Si volvemos a nuestro ejemplo, suponiendo que lo hemos programado todo bien y **tomando como pivote el primer elemento de la lista** (para no andar en el ejemplo calculando la posición media), tras los pasos **1 y 2**, selección del pivote y recolocación de elementos, tenemos lo siguiente:

Original:	03	07	11	02	09	01	08	05	10	06	04
Pivote											
Primer paso:	01	02	03	09	11	07	08	05	10	06	04
	*****			*****							

El primer elemento, el elegido como pivote, el “03”, ha quedado efectivamente colocado en su lugar definitivo y ahora tenemos dos conjuntos de elementos, los menores que “03” a su izquierda, y los mayores a su derecha. Ambos conjuntos están en principio desordenados. Se puede observar que el de la izquierda ya está bien, pero no hay forma de que el algoritmo pueda saberlo todavía. Ahora hay “simplemente” que aplicar la misma medicina a los dos conjuntos de elementos: primero al uno y luego al otro. ¿Hasta cuando?

Pues hasta que el subconjunto que haya que clasificar esté *necesariamente ordenado*... lo que ocurre cuando tiene tamaño cero o uno, es decir, cuando hay uno o ningún elemento a clasificar.

No os extrañe que hable de “cero elementos”, pues es perfectamente posible: si en el ejemplo hubiésemos tomado como pivote al elemento central (o sea, el que está *físicamente en medio* de la lista), el valor elegido hubiera sido el “01”, que es el que está en la posición central de la tabla, la sexta... elemento que resulta que es el de clave más pequeña y, por tanto, hubiera quedado ubicado en el extremo izquierdo de la lista, con lo que no hubiera tenido ningún elemento a la izquierda y todos los demás, desordenados, a la derecha.

En definitiva, si el tamaño del subconjunto a ordenar es menor que dos, no ejecutamos esa parte, garantizando que el algoritmo termina. Es más, podemos mejorar significativamente el rendimiento del algoritmo si, cuando el tamaño del subconjunto es exactamente dos, en lugar de volver a llamar recursivamente al algoritmo nos limitamos a comprobar si estos dos elementos contiguos están bien ordenados, en cuyo caso los dejamos como están, o mal ordenados, en cuyo caso los intercambiamos, evitando así una relativamente costosa re-llamada recursiva al algoritmo para ordenar una lista de tan sólo dos elementos.

Sigamos, pues, con nuestro ejemplo:

El subconjunto de la izquierda, “01”, “02”, está ya clasificado. Como tiene dos elementos, hemos comprobado si estaban ordenados entre sí, y como sí que lo están, los mantenemos sin cambio. Tratamos entonces sólo el segundo subconjunto. Elegimos el pivote (el primer elemento, hemos dicho: el “09”, pues), e intercambiamos elementos nuevamente para mover los menores a la izquierda y los mayores a la derecha.





El tramo de elementos a la derecha del pivote (“10”, “11”) sólo tiene dos elementos, luego los ordenamos entre sí si no lo estuvieran ya, que sí lo están, y nos dedicamos ahora al subconjunto izquierdo, los cinco elementos que siguen sin sombrear. Y así sucesivamente.

Casi se ve a simple vista que **es un procedimiento muchísimo más rápido que la burbuja con todas sus variantes...** aunque es también bastante más complicado de programar correctamente. En general, salvo casos especialmente diseñados para fastidiarle, obligándole a tomar siempre como pivote al peor de los elementos posibles, el número de elementos a revisar, su complejidad, pues, es del orden de  $n \cdot \log n$ , es decir, el logaritmo en base 2 del número  $n$  de elementos de la lista.

Para una lista de cien mil elementos, la burbuja necesitará revisar del orden de diez mil millones de elementos para clasificarla... con las diferentes mejoras, quizá lo reduzcamos a *sólo* mil o dos mil millones. Quicksort sólo necesitará revisar del orden de 1,5 a 2 millones de elementos para hacer el mismo trabajo. Bastante menos y, además, la diferencia se hace mayor cuanto mayor sea el tamaño de la lista.

El punto “débil” de Quicksort es *la elección del pivote*. Si la elección es sistemáticamente mala, es decir, el pivote elegido es de los primeros o de los últimos de la lista ordenada, pero no alguna vez, sino siempre, siempre, su rendimiento puede ser mucho peor de lo esperado. En cambio, si, gracias a la acción de algún sortilegio, elegimos siempre como pivote el elemento que ocupará exactamente la posición central de la lista definitiva, su rendimiento será óptimo.

Muchas implementaciones de Quicksort aplican diversas estrategias para optimizar en lo posible esta elección, como tomar el primer elemento, el último y el central, y hallar la media de sus valores, *si se trata de números, claro, porque si son nombres... no sé yo cuál será la media de “Sánchez” y “Pérez”, no sé... ¿quizá “Rodríguez”?*

Otras, directamente se revisan la lista para asegurarse que toman el mejor valor posible, el que, al final de la partición, quedará exactamente en el punto medio de la lista... con un proceso que es también costoso, así que muchas veces no compensa.

Y otras utilizan una solución más de andar por casa (que es la que a este informático viejo y paleta más le gusta) que es simplemente **tomar como pivote el elemento central de la lista** (si la lista tiene, por ejemplo, 100 elementos, será el que ocupe la posición  $(1+100)/2 = 50$ ).

Esta suposición puramente empírica se basa en que las listas del mundo real no son casi nunca aleatorias, sino que suelen tener algún tipo intrínseco de ordenación o cuasi-ordenación. Según la Ley de los Grandes Números es bastante eficiente usar esta estrategia, pues en un número significativo de casos la posición final del pivote elegido no estará muy alejada de su posición inicial. No es nada científico, como veis. Lo que sí puedo asegurar es que, en el mundo real, esta sencilla asunción funciona de maravilla.

Quicksort es, efectivamente, un algoritmo relativamente complejo, pero también es uno de los algoritmos más estudiados, trabajados y optimizados por informáticos de todo el mundo, que han conseguido finalmente reducir el código necesario para su implementación hasta extremos... bueno, *hasta el extremo*. A continuación tenemos tres ejemplos de esta reducción desaforada de código:

En **Python** (autor: Nathan Gray)

```
def qsort(L):
    if len(L) <= 1: return L
    return qsort( [ lt for lt in L[1:] if lt < L[0] ] ) +
        [ L[0] ] + qsort( [ ge for ge in L[1:] if ge >= L[0] ] )
```

En **C** (autor: Harry Hardjono)

```
void quicksort(int lo, int hi) {
    int i, p;
    if (lo >= hi) return;
    for (p = (i = lo) - 1; i <= hi; i++) if (a[i] <= a[hi] && i != ++p)
        a[i] ^ a[p] ^ a[i] ^ a[p];
    quicksort(lo, p - 1);
    quicksort(p + 1, hi);
}
```

En **Haskell** (autor: bayareaguy)

```
quicksort (x:xs) =
    quicksort [ i | i <- xs, i < x ]
    ++ [x] ++
    quicksort [ i | i <- xs, i >= x ]
```

```
quicksort [] = []
```

Sí, definitivamente son algoritmos bastante “*oscuros*”. No me preguntéis si funcionan o no, no tengo ni idea, aunque me da en la nariz que no funcionan *del todo*: puro instinto informático, desarrollado con los años. Pero ya veis, hay quien es capaz de codificar el algoritmo en sólo tres o cuatro líneas, luego... *¡no será tan difícil!*

Aunque yo, que lo mío es el Assembler y el Cobol (igual no se había notado), necesito *bastante más de tres líneas* para codificarlo...

A continuación, y para no ser menos, os dejo un pequeño regalo que quizá alguno de vosotros agradezca (aunque no muchos, me temo): **una implementación, muy cercana a la más eficaz, del algoritmo Quicksort... en Cobol**. Está basado en la implementación de Robert Sedgewick, con alguna mejora. Aseguro que funciona. *Palabrita*. Siempre, en todos los casos, excepto cuando la lista a ordenar está vacía, pero eso es porque con la edad me he vuelto vago y no me he molestado en poner el *IF* inicial correspondiente... y además, si la lista está vacía, ¿qué vas a ordenar, alma de cántaro?

Se trata de un módulo (ésta es la forma correcta de implementarlo), de nombre QUIKSCBL, que necesita ser llamado por un programa principal entregándole como parámetros la tabla real a clasificar y el número real de elementos que tiene, y devuelve la tabla clasificada en el mismo área de memoria de la tabla original. En el programa la tabla a

clasificar tiene 10.000 elementos de diez posiciones cada uno, y se clasifica por la clave compuesta por los 5 caracteres iniciales de cada elemento. Naturalmente, cada implementación deberá ajustar estos valores para que se correspondan con los datos reales del caso.

Como QuickSort es un algoritmo recursivo, en esta implementación la recursividad se soluciona con una pila (un Stack, un LIFO) que lleva el control de cada pasada.

Y, por fin, le he puesto comentarios para ayudar a los menos experimentados con el Cobol (o sea, casi todos) a comprender para qué sirve cada bloque de instrucciones, aunque el hecho de que las instrucciones Cobol se escriben en un casi perfecto inglés ayuda bastante a entenderlas.

Espero que sea útil. Para algunos, al menos.

#### **IDENTIFICATION DIVISION.**

##### **PROGRAM-ID. QUIKSCBL.**

- \* **Módulo que clasifica TABLA-A-CLASIFICAR, por CLAVE**
- \* **(en este ejemplo, sus cinco primeras posiciones).**
- \* **En NUM-ELEMENTOS recibe el número real de elementos de la**
- \* **tabla a clasificar. Debe ser mayor que cero;**
- \* **si puede tener valor cero hay que preguntarlo para**
- \* **terminar inmediatamente el proceso.**
- \*

#### **DATA DIVISION.**

##### **WORKING-STORAGE SECTION.**

**01 CLAVE-MEDIO PIC X(5).**

\*

**01 ELEM-INTERCAMBIO PIC X(10).**

\*

**01 INI-WORK PIC 9(8) COMP.**

**01 FIN-WORK PIC 9(8) COMP.**

**01 MEDIO PIC 9(8) COMP.**

**\*\*\*\* PILA (para resolver la recursividad de "PARTICION").**

**01 STACK-RECURSIVIDAD.**

**05 ELEM-STACK OCCURS 100 INDEXED BY IND-STACK.**

**10 INISTACK INDEX.**

**10 FINSTACK INDEX.**

**\*\*\*\*\***

#### **LINKAGE SECTION.**

**01 NUM-ELEMENTOS PIC 9(8) COMP.**

\*

**01 TABLA-A-CLASIFICAR.**

**05 ELEMENTO OCCURS 10000, INDEXED BY INICIAL-P,  
FINAL-P,  
INI,  
FIN,  
IND-WORK.**

**10 CLAVE PIC X(5).**

**10 FILLER PIC X(5).**

\*\*\*\*\*

**PROCEDURE DIVISION USING NUM-ELEMENTOS,  
TABLA-A-CLASIFICAR.**

**MAIN SECTION.**

**COMIENZO.**

- \* Inicialización de la Pila de Recursividad e índices
- \* de la tabla a clasificar.
- \*

SET INICIAL-P TO 1.  
SET FINAL-P TO NUM-ELEMENTOS.  
SET IND-STACK TO 2.  
SET INISTACK (2) TO INICIAL-P  
SET FINSTACK (2) TO FINAL-P.

- \* Proceso principal recursivo de QUICKSORT.  
PERFORM PROCESAR-PARTICION  
UNTIL IND-STACK LESS 2.  
EXIT PROGRAM.
- \*

**PROCESAR-PARTICION.**

- \* Recuperación de la Cabeza de la Pila, con datos de la
- \* llamada recursiva: índices Inicial y Final de la,
- \* partición a tratar y llamada a PARTICION para dividirla
- \* en dos subconjuntos.
- \* Descabezar la Pila y comprobar los dos Subconjuntos:
- \* Clave Menor o Igual que el Pivote: INI <--> FINAL-P);
- \* Clave Mayor o Igual que el Pivote: (INICIAL-P <--> FIN).
- \* Si un Subconjunto tiene menos de dos elementos ya está
- \* ordenado. Si tiene dos se comprueba si están ya en orden
- \* y se intercambian si es preciso. En otro caso se insertan
- \* sus datos en la Pila para próximas llamadas recursivas.
- \*

SET INICIAL-P TO INISTACK (IND-STACK).  
SET FINAL-P TO FINSTACK (IND-STACK).  
PERFORM PARTICION.  
SET IND-STACK DOWN BY 1.

- \*
- \* SUBCONJUNTO 1: Delimitado por INI Y FINAL-P.  
SET IND-WORK TO INI  
SET IND-WORK UP BY 1  
IF FINAL-P GREATER IND-WORK  
SET IND-STACK UP BY 1  
SET INISTACK (IND-STACK) TO INI  
SET FINSTACK (IND-STACK) TO FINAL-P  
ELSE  
IF FINAL-P EQUAL IND-WORK AND  
CLAVE (INI) GREATER CLAVE (FINAL-P)  
MOVE ELEMENTO (INI) TO ELEM-INTERCAMBIO

**MOVE ELEMENTO (FINAL-P) TO ELEMENTO (INI)**  
**MOVE ELEM-INTERCAMBIO TO ELEMENTO (FINAL-P).**

**\***

**\* SUBCONJUNTO 2: Delimitado por INICIAL-P Y FIN.**

**SET IND-WORK TO INICIAL-P**

**SET IND-WORK UP BY 1**

**IF FIN GREATER IND-WORK**

**SET IND-STACK UP BY 1**

**SET INISTACK (IND-STACK) TO INICIAL-P**

**SET FINSTACK (IND-STACK) TO FIN**

**ELSE**

**IF FIN EQUAL IND-WORK AND**

**CLAVE (INICIAL-P) GREATER CLAVE (FIN)**

**MOVE ELEMENTO (INICIAL-P) TO ELEM-INTERCAMBIO**

**MOVE ELEMENTO (FIN) TO ELEMENTO (INICIAL-P)**

**MOVE ELEM-INTERCAMBIO TO ELEMENTO (FIN).**

**\***

**\*\*\*\*\***

**\* Partición del Subconjunto (INICIAL-P <--> FINAL-P).**

**\* Se selecciona el Pivote (el elemento central), y se**

**\* almacena su CLAVE en CLAVE-MEDIO.**

**\***

**PARTICION SECTION.**

**PROCESO.**

**SET INI TO INICIAL-P.**

**SET FIN TO FINAL-P.**

**SET INI-WORK TO INI**

**SET FIN-WORK TO FIN**

**COMPUTE MEDIO = (INI-WORK + FIN-WORK) / 2.**

**MOVE CLAVE (MEDIO) TO CLAVE-MEDIO.**

**\***

**PERFORM INTERCAMBIAR-ELEMS**

**UNTIL INI GREATER FIN.**

**\***

**GO TO FINAL-PARTICION.**

**\***

**INTERCAMBIAR-ELEMS.**

**\* Intercambio de elementos. A la vez por el principio y**

**\* el final del Subconjunto; cada vez que se encuentran dos**

**\* elementos descolocados, se intercambian hasta que los dos**

**\* índices de intercambio (INI y FIN) se cruzan.**

**\***

**\* 1) Búsqueda de elementos descolocados en el principio del**

**\* Subconjunto (su CLAVE es mayor o igual que CLAVE-MEDIO).**

**\* 2) Búsqueda de un elemento descolocado en el final del**

**\* Subconjunto (su CLAVE es menor o igual que CLAVE-MEDIO).**

**\* 3) Intercambio de ambos elementos (excepto si los índices**

```

* INI y FIN se han cruzado, en cuyo caso se ha terminado).
*
  PERFORM VARYING INI FROM INI BY 1
    UNTIL CLAVE (INI) NOT LESS CLAVE-MEDIO
  END-PERFORM.
*
  PERFORM VARYING FIN FROM FIN BY -1
    UNTIL CLAVE (FIN) NOT GREATER CLAVE-MEDIO
  END-PERFORM.
*
  IF INI NOT GREATER FIN
    MOVE ELEMENTO (INI) TO ELEM-INTERCAMBIO
    MOVE ELEMENTO (FIN) TO ELEMENTO (INI)
    MOVE ELEM-INTERCAMBIO TO ELEMENTO (FIN)
    SET INI UP BY 1.
    SET FIN DOWN BY 1.
  *
  FINAL-PARTICION. EXIT.
  *****

```

...Y aquí termina este capítulo dedicado a la Ordenación...

¿...?

**No**, claro que no. Porque estos algoritmos ordenan listas en la memoria del ordenador, que tiene la fea costumbre de ser de limitado tamaño... así que ¿qué ocurre si deseamos clasificar una lista de varios millones de elementos que, en definitiva, **no cabe en la memoria**? Traducción al lenguaje de los años setenta, con la escasísima memoria de los ordenadores de entonces: ¿Cómo se clasificaban *todas* las listas, por pequeñas que sean?

Pues hay que recurrir al **Ordenamiento Externo**, es decir, usar dispositivos de memoria externa para realizar la clasificación.

Ahora se usa siempre disco magnético para estos menesteres, pero yo he llegado a utilizar el **Sort en Cinta Magnética** en mis años mozos: con 32 Kb de memoria, no había mucho espacio para ordenar nada en memoria, como es evidente, y con discos de 4 Mb, en muchos casos no era tampoco factible usarlos como espacio de trabajo. Incluso antes que eso, se clasificaban los ficheros en tarjeta perforada, mediante unas máquinas especializadas llamadas “Tabuladoras” (Tabulating Machines).

La primera, como ya vimos, fue la que construyó Hollerith para el dichoso censo estadounidense de 1890; con el paso del tiempo llegaron a una gran perfección, tanto, que fue en el negocio de las tabuladoras de fichas perforadas donde IBM fundó su crecimiento futuro.

Estas máquinas tomaban un taco grande, de decenas o centenares de miles de tarjetas, las leían y las iban clasificando en diferentes cajetines por el campo que se definiera.

Cuando acababa el primer paso del proceso y se tenían todas las tarjetas iniciales separadas en montoncitos en los cajetines, se iba haciendo el mismo trabajo con el contenido de cada uno de los montones, hasta que quedaba clasificado el fichero completo.

Vale, es cierto, tardaba mucho, era bastante manual y propenso a errores... pero mucho menos que hacerlo totalmente a mano, eso, sin duda.

*Atención: **Batallita**.* Dije que en este capítulo no iba a haber ninguna, ¿verdad? Pues no: también habrá.

Recuerdo que cuando yo era muy, muy joven, es decir, un niño (debía ser a principios de los sesenta, ¿1961, 1963?... por ahí) había un Concurso en la Tele (en directo, claro, *TODO* era en directo entonces; y no preguntéis *en qué Tele*, **en la Tele** y punto), de cuyo nombre no me acuerdo, donde cantaban ciertos artistas de la época de los que, obviamente, tampoco me acuerdo... aunque fijo, fijo que al menos Marisol, José Guardiola, Raphael y Manolo Escobar pasarían por allí...

Los televidentes, los pocos que había entonces, pues la Tele, lo que ahora se llama “Televisión Española”, no comenzó a emitir en España hasta finales de 1956, compraban durante la semana siguiente a la emisión del concurso, creo recordar que en los Estancos, unas tarjetas preimpresas y sin perforar, donde esos televidentes/concursantes agujereaban de no sé qué forma, seguramente con lápiz (los más avanzados, bolígrafo), cuáles habían sido los cantantes que más les habían gustado y las enviaban luego por correo (postal, claro) a *la Tele*.

Entonces, todas las tarjetas recibidas agujereadas de aquella manera, con lo que vaya Vd. a saber lo que de verdad habría allí perforado, se pasaban por una tabuladora de aspecto imponente, o al menos eso nos parecía a nosotros, los atónitos teleespectadores de la época, durante el programa siguiente, en riguroso directo, claro, y de allí surgía, por arte de *tecnológica magia*, el taco de las tarjetas que habían acertado quién ganaría, o quién cantó mejor o lo que fuera, y los acertantes se repartían finalmente un premio de algunos miles de pesetillas para todos. Que entonces era un buen dinero...

La tabuladora que utilizaban, que creo recordar que no sólo era de IBM, sino que estaba físicamente ubicada en las oficinas de IBM en Madrid, tenía un aspecto similar a la de la siguiente ilustración, o al menos por algo así la tengo en mis infantiles recuerdos. Se trata de una tabuladora nada menos que de los años 40 (Wikipedia, Sandstein, CC by SA 3.0).



Este programa representó un auténtico empujón hacia la modernidad para nuestro país y para la época: efectivamente, fue la primera vez que muchos, entre otros yo mismo, se enteraron de que existían estos artefactos y para qué valían... y además fue el auténtico precursor de los hoy inevitables concursos de envío de SMS's... **¡SMS's de cartón y por Correo Certificado**, eso sí que es friki!

*Lo siento, sé que prometí que este capítulo no iba a tener batallitas, pero es que no he podido resistirme...*

Volviendo al Sort, para clasificar un fichero mediante Ordenamiento Externo, el sistema que se usa se basa en **utilizar toda la memoria interna posible** (por escasa que ésta sea), **llenarla con la primera parte de la lista original** (todo lo que quepa), **ordenarla en memoria y copiarla, una vez ordenada, a la memoria externa** (en diferentes ficheros temporales, alternándolos), e ir realizando esta función hasta terminar la lista.

Entonces se comienzan a leer coordinadamente los ficheros temporales, por bloques, los mismos que han sido ordenados en memoria, generando nuevos ficheros temporales, que tendrán bloques más grandes... hasta que, en una última pasada, se obtiene el fichero definitivo clasificado. Para hacer este proceso se utiliza un *Merge*, ¿recordáis? No deja de ser parecido a lo que hacía la primigenia máquina de Hollerith, aunque sin tarjetas. Tardar, tarda más que hacerlo en memoria, claro. Muchísimo más. Pero, cuando no queda más remedio...



Veamos cómo funcionaría un ordenamiento externo con un ejemplo.

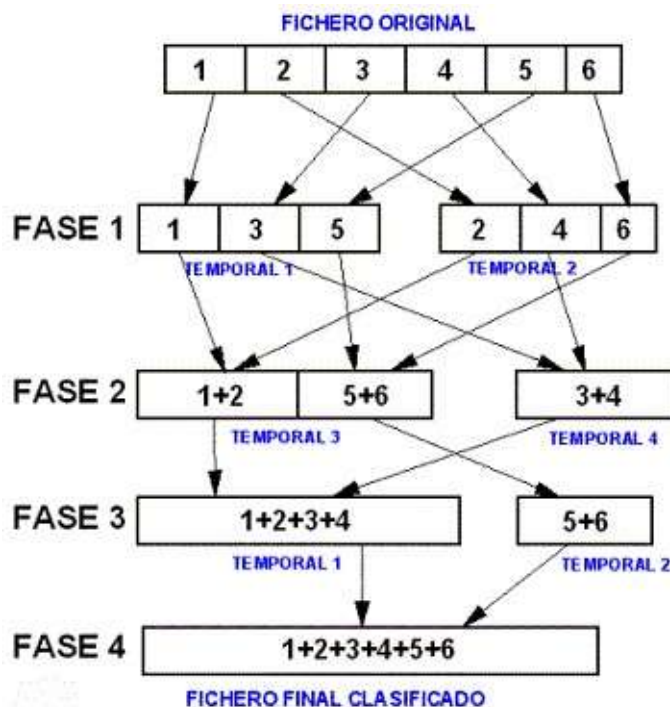
Supongamos que queremos ordenar un fichero que tiene, digamos, 550.000 registros, pero en nuestra memoria interna disponible sólo nos caben 100.000... Como no podemos hacer la ordenación directamente en memoria, pues no tenemos suficiente, hay que recurrir a una **Clasificación Externa**.

Necesitaremos espacio temporal adicional, así que supongamos que definimos cuatro ficheros temporales para su uso por el algoritmo. Cuantos más ficheros definamos, más eficiente será el proceso, como fácilmente podremos ver en unos momentos. Estos ficheros temporales son puramente secuenciales, por lo que podrían estar en disco, en cinta magnética, incluso en tarjeta perforada, si encontramos...

Lo primero que hacemos (**FASE 1**) es ir leyendo bloques de 100.000 registros (que son los que, como hemos dicho, nos caben en la memoria) y, una vez allí, ordenarlos por un algoritmo rápido. *QuickSort* o *HeapSort*, o alguna variación de ellos, suelen ser los preferidos para estos menesteres.

Una vez ordenados, los escribimos en los ficheros temporales 1 y 2, alternativamente.

En la ilustración se puede observar un diagrama esquemático de cómo funcionaría el proceso.



En el fichero original los registros están completamente desordenados. Podemos ver cómo los seis bloques en que se divide el fichero, 5 de 100.000 registros y, el último, de 50.000, una vez clasificados internamente van a parar, los bloques 1, 3 y 5, al fichero *temporal 1*, y los bloques 2, 4, y 6, al *temporal 2*. Estos bloques 1 a 6 ahora sí están ordenados, pues se han clasificado sus registros en memoria.

Bien, a partir de ahora ya no usamos ningún algoritmo de ordenación, sino sólo de mezclado: el **Merge**, del que ya hemos hablado antes. Efectivamente, en **FASE 2** leemos

coordinadamente los dos primeros bloques (1 y 2), de ambos ficheros, generando por tanto un bloque ordenado de 200.000 registros, que escribimos en el fichero *temporal 3*.

Luego hacemos lo mismo con los bloques 3 y 4, pero grabando ahora en el *temporal 4*, y luego los 5 y 6, que son grabados en el *temporal 3* nuevamente.

En la **FASE 3** leemos los ficheros temporales recién creados (*temporal3* y *temporal4*) y usamos de nuevo los temporales 1 y 2, que han quedado libres; ahora hacemos *merge* sobre bloques de 200.000 registros ya clasificados; primero se mezclan los bloques 1+2 y 3+4, grabando el resultado, ya ordenado, al *temporal1*. El bloque 5+6 se graba tal cual en el *temporal2*, puesto que no quedan ya bloques en el *temporal4* con los que mezclarse.

Por fin, la **FASE 4** vuelve a mezclar los bloques de 400.000 registros de los temporales 1 y 2, generando, en este caso, el fichero final ya clasificado. Si tuviéramos más registros, en FASE 5 los bloques serían ya de 800.000 registros, en FASE 6, de 1.600.000, etc.

En definitiva, hemos usado un algoritmo “**Mergesort**”, que es muy eficiente y, además, paralelizable, aunque tiene una pega: necesita memoria adicional, al menos tanta como la que ocupe la lista original a ordenar (mejor, el doble, como en el ejemplo, para no machacar el soporte con el fichero original, *no vaya a cascar a mitad del trabajo...*). Es decir, **Mergesort es el algoritmo de referencia cuando el ordenamiento ha de hacerse en memoria externa.**

Como podéis imaginar, este algoritmo básico es susceptible de ser mejorado bastante. Por ejemplo, usando seis ficheros temporales en vez de cuatro podría hacerse el mismo trabajo en sólo tres Fases, no en cuatro. También, en la Fase 3, no sería necesario grabar los bloques 5+6 en el fichero temporal 2, sino que la Fase 4 podría mezclar el temporal 1 con el temporal 3, a partir del punto exacto donde se quedó... y otras muchas que se os ocurrirán. Pero vamos a dejarlo aquí, en aras a la (*ya inexistente*) brevedad del capítulo.

Espero haber sido instructivo a la par que ameno. O, al menos, que os hayáis enterado de algo...

En el próximo capítulo volveremos de nuevo a las batallitas, esta vez para recordar cómo comenzaron los cambios en la informática que nos llevaron de cabeza a la que conocemos en el Siglo XXI.

## 9 - La irresistible irrupción del PC en la Empresa

Este capítulo trata de uno de los aspectos del cambio tecnológico que se produjo durante los años 80, del que quizá lo más destacado, interesante y con mayor relevancia posterior fue la adopción del PC de manera generalizada, en la empresa, primero, y poco después, en todas partes...

Lo dije en capítulos anteriores y lo repito: no esperéis un resumen periodístico ni enciclopédico al estilo de la Wikipedia. Contaré más bien mis experiencias personales al respecto, lo que se oía y pensaba en aquellos años por los que estábamos en la *pomada*, no la “Historia oficial”, que suelen escribir siempre los ganadores... inveterada tradición de los humanos desde los antiguos egipcios, que ya cambiaban las inscripciones en piedra para que siempre estuvieran de acuerdo con la “*postura oficial*” del faraón de turno: fueron ellos quienes en realidad inventaron el “*Ministerio de la Verdad*” inmortalizado por George Orwell en su obra maestra “1984”.

Ahora bien, teniendo en cuenta que en aquellos años estuve, no sólo por razones personales, sino también profesionales, muy encima de los cambios tecnológicos que ocurrían (que, además, no eran tan rápidos como ahora, donde es imposible enterarse de todas las novedades con días de tan sólo 24 horas), y que debí ser de los primeritos de España que tuvieron un PC entero para mí solo encima de la mesa de la oficina (llevo treinta años siendo usuario de uno), es posible que estas experiencias personales puedan tener algún interés para los aficionados a la arqueología informática...

Como comenté en el capítulo dedicado al método de trabajo de los ochenta, a mediados de esa década volví a cambiarme a otro Banco. Lo que no os dije es que, en buena parte, el cambio fue obligado, bueno, quizá no “obligado”, tan sólo *sugerido*, *facilitado*, *animado* por el hecho de que aquel en el que estuve hasta entonces entró en dificultades.

Fue uno de los muchos damnificados por la “Crisis bancaria de los Ochenta”, y sobre él se cernieron muy negros nubarrones... Los que leáis prensa, oigáis radio o veáis televisión, igual os suena de algo qué es eso de “*Crisis Bancaria*”, rescate de bancos y demás zarandajas...

Así que, ante un futuro incierto en esta entidad, aunque luego no lo fue tanto, me puse en contacto con antiguos colegas de la Universidad y tuve la oportunidad de recibir una magnífica oferta para incorporarme a otro Banco, en un puesto de mayor responsabilidad... y retribución, naturalmente. Me *monté en el dólar*, vaya. Lástima que después me haya bajado... *dos veces*.

Más tarde volveré a hablar de la situación en la Banca, que es la que me conozco bien, en aquellos años, pero ahora me voy a centrar en describir el **panorama de la tecnología informática en aquellos años, primera mitad de la década de los 80.**

La informática “tradicional”, es decir, la gran informática empresarial o gubernamental, sobre todo la dedicada a la gestión, es decir, la de los mainframes, había llegado a un punto de desarrollo importante.

Todas las grandes empresas de todos los sectores, ministerios y demás tenían ya su

mainframe funcionando y se estaban escribiendo muchas aplicaciones nuevas, bien para sustituir las que habían quedado obsoletas, bien para solucionar nuevas necesidades, sobre todo derivadas de los emergentes procesos online.

Es decir, *las grandes empresas tenían ya su mainframe*. Varias marcas ofertaban grandes ordenadores: NCR, BULL, Siemens, Control Data, BURROUGHS, UNIVAC... Estas dos últimas se fusionaron en 1986 para dar lugar a UNISYS.

Aunque todas estas marcas ofrecían ordenadores de muy buena calidad y vendieron bastantes sistemas en las dos décadas anteriores, en los ochenta la *carrera del mainframe* ya tenía un claro ganador: **IBM**. La serie 303x, sustituta del mítico 370, había tenido ya un gran éxito, pero el lanzamiento de la serie **IBM 308x** y, poco después, el de la serie IBM 3090, literalmente arrasó a la competencia, que poco a poco dejó libre el campo de batalla.

De hecho, a partir de la segunda mitad de los años 80 la principal competencia de IBM en mainframes fueron, curiosamente, máquinas de otros fabricantes (Fujitsu, Amdahl, Hitachi...) que, producto de la ingeniería inversa, eran totalmente compatibles con las de IBM... y mucho más baratas, desde luego.

Funcionaban, además, con su mismo software, que ya os describí someramente en los capítulos dedicados a los mainframes de IBM y al método de trabajo que utilizábamos en los 80: MVS, VSAM, VTAM, IMS/DB, IMS/DC, CICS, etc.

Estos eran los clónicos del momento, conocidos como **PCM** (*Plug Compatible Machines*), y tuvieron su momento de gloria a raíz de una sentencia del Tribunal Antitrust de EEUU que obligó a IBM a no negarse a vender o alquilar su software a los clientes que tuvieran este tipo de máquinas. Además, empezaron a venderse dispositivos periféricos (discos, cintas magnéticas, impresoras, pantallas, etc.) compatibles también con los de IBM, pero de otras marcas. Para IBM, evitar estas ventas de periféricos clónicos era mucho más difícil de frenar que las de las propias CPU's.

IBM reaccionó también, naturalmente, incluyendo cada vez más microcódigo (*firmware*) en sus ordenadores, de tal manera que fuera más difícil para la competencia descifrar su funcionamiento mediante ingeniería inversa... pero esta estrategia, en realidad, le hizo ganar apenas unos meses de tranquilidad.

En cualquier caso, los mainframes de IBM eran esos años las máquinas más vendidas, y no digamos ya su software, que *había alcanzado una posición de virtual monopolio en este mercado*.

**Pero IBM tenía un problema con esto:** No hay *tantas* grandes empresas o instituciones en el mundo. La mayoría tenían ya, o tendrían en unos pocos años, un gran mainframe suyo o quizá un PCM con software de IBM, sistema que habría que ir ampliando con el tiempo, irlo sustituyendo por otro más moderno en su debido momento y pagar religiosamente cada año el alquiler del software. Todo esto reportaba pingües ingresos a IBM (de hecho lo sigue haciendo: la División de Mainframes sigue siendo de las más rentables de IBM en la actualidad), *pero no bastaba*.

**IBM quería más.** De hecho, siguiendo la más estricta lógica empresarial, **IBM lo quería todo**. Como veis, cuando se acusa a según qué Compañía actual de tecnología justamente de la misma cosa... no hay nada nuevo bajo el sol.

En el mercado de los pequeños ordenadores, los “minis”, adecuados a pequeñas empresas que no necesitaban tanta potencia como para poder amortizar un mainframe, en entornos ingenieriles, etc., IBM tenía también líneas de productos específicos: los de la

Serie/1 con su fastuoso floppy de 8 pulgadas, y el System/3 y sus múltiples sucesores en el tiempo: System/32, System/34, System/36 y System/38, que dieron lugar al final, en 1988 al AS400 que desde hace pocos años ha sido rebautizado como iSeries. Caray con los cambios *marketinianos* de nombre...

Todas estas máquinas, unas herederas de las otras, son el reino del RPG, acrónimo de “*Report Program Generator*”, y luego de su sucesor, el RPG II, aunque también tienen compiladores para otros lenguajes, como Cobol desde hace muchos años, así como C, etc.

Su arquitectura actual los hace tan robustos y seguros como los mainframes, más pequeños y menos potentes, menos complicados de administrar y mucho más baratos; siguen siendo sistemas con un mercado significativo, sobre todo en las empresas medianas.

Pero hasta 1990 no tuvo IBM en el mercado una gama de ordenadores basados en UNIX: los RS/6000, por **RISC Systems**, ya que los procesadores que utilizaron eran de tecnología RISC: el “IBM Power”, el primer procesador RISC utilizado comercialmente de forma masiva. Su Sistema Operativo es AIX, la versión propietaria de IBM de UNIX, del mismo modo que Solaris es la de Sun, o HP/UX la de Hewlett-Packard: ya sabéis: hay diez o quince UNIX todos iguales... y todos distintos. La gama de Sistemas UNIX (bueno, quería decir AIX), los RS/6000, continúa siendo parte fundamental de la oferta de IBM en la actualidad, y más desde que vendió su división de PC’s a Lenovo.

Todos estos ordenadores, así como todos sus dispositivos, eran de tecnología propia: chips propios y específicos, periféricos específicos también de IBM, software propio e incluso, en las gamas S/3x, tenían también lenguaje de programación propio: el RPG. Es decir, seguían la misma técnica que había utilizado con éxito durante años la División de mainframes: **usar exclusivamente tecnología propia bajo patente de IBM**, sacando así el máximo partido a los excelentes Centros Tecnológicos que IBM tenía repartidos por todo el mundo, incluida España.

Pero de la misma manera que sus mainframes habían triunfado de forma definitiva, **en el área de las máquinas pequeñas IBM no triunfó en absoluto**. Vendieron muchos sistemas, desde luego, puesto que su fuerza comercial era realmente poderosa, pero aquí la competencia era feroz por parte de muchos otros fabricantes de hardware y software que disponían también de excelentes productos: Digital Equipment Corporation, Olivetti, Nixdorf, Philips, Siemens, Hewlett-Packard, Bull, etc.

Cubrían entre todos una amplia gama de necesidades, bien utilizando sistemas operativos propios, como el VMS del Sistema VAX de Digital, o bien usando alguna versión del UNIX de la época.

Entre tanto fabricante, IBM era únicamente un competidor más, un *lobo* más en la *jauría*, y seguramente ni siquiera el más exitoso de todos en este segmento de mercado... seguramente lo fuera Digital Equipment en aquellos años, aunque quizás me equivoque.

En cuanto a los terminales especializados, es decir, el terminal financiero, el terminal punto de venta, etc., IBM tenía, cómo no, oferta en todos estos campos y, cómo no, con tecnología propia: el Sistema Financiero IBM 3600, del que hablábamos en el capítulo dedicado al método de trabajo de los 80, para bancos y entidades financieras; para comercios, el Sistema de Terminal Punto de Venta IBM 3650, y para otras áreas de negocio tenía también una serie de equipos especializados que no voy a detallar, pues resultaría tedioso: la realidad era que IBM estaba presente y era uno de los competidores de referencia en casi todas las áreas de negocio habidas y por haber.

Pero no sólo toda la tecnología que IBM ofrecía era propia, sino que era *casi incompatible entre sí*: no había prácticamente ninguna transferencia entre la tecnología de los mainframes y la de los terminales financieros, o la del Sistema/34, por ejemplo. Cada sistema se desarrollaba en un laboratorio diferente y, por lo visto, debían estar todos ellos *peleados* entre sí, aunque en IBM siempre dijeron lo contrario... *¡faltaría más!*

Debido a esto, o quizás a pesar de esto, no lo sé, también en estas áreas IBM afrontaba una dura competencia de otros fabricantes que, de hecho, vendían mucho y bien, incluso para conectarse a los inevitables ordenadores centrales de IBM. Por ejemplo, Philips, Nixdorf y Olivetti, en el mercado europeo sobre todo, tenían excelentes Sistemas de Terminal Punto de Venta para tiendas minoristas y Sistemas Financieros tan eficaces como los propios de IBM, y más baratos.

En una palabra, **tampoco llevaba IBM en este segmento especializado la voz cantante.**

Así las cosas, estaba comenzando a aparecer un nuevo segmento de mercado hasta hacía muy poco tiempo inexistente: El del **Ordenador Doméstico.**

El pionero en este campo fue sin duda **Apple**, con su **Apple I** de 1976, fabricado a mano sobre pedido, en total unas 200 unidades, que se entregaba en forma de kit, con una placa base, la CPU, la memoria y un chip de video, de sólo texto, para que el usuario lo montase, y sobre todo con su **Apple II**, lanzado en 1977, que se fabricó ya de forma industrial y que se vendía ya completo, con su carcasa, su teclado y todo.

En la ilustración, un Apple II de 1978.



Eran máquinas muy poco potentes, con sólo unos pocos Kb de RAM, lector/grabador de cinta magnética y sin disco (aunque pronto se escribió el driver para manejar un disco flexible en el Apple II; el que aparece en la foto tiene ya, como puede verse, dos disqueteras de 5,25").

En cuanto a pantalla, aunque tenían la suya propia eran también capaces de conectarse a un aparato de televisión cualquiera, lo que permitía abaratar los costes.

Eran máquinas baratas, no con criterios actuales, desde luego, pero sí comparadas con cualquier alternativa existente en la época: el precio de la configuración más cara, la que llevaba nada menos que 48KB de memoria, fue inicialmente de alrededor de 2.600 dólares. Ojo: 2.600 dólares de hace treinta años... ¡y eso era mucho dinero para un particular! Sin embargo, a pesar de eso, estos incipientes ordenadores domésticos se usaban sobre todo, no nos engañemos, *para jugar*. Por gente adinerada, eso sí, pero para jugar, debido a que la escasísima capacidad de almacenamiento, la dificultad en recuperar y

guardar datos externos y, por qué no decirlo, su poca fiabilidad los hacían poco menos que inviables para cualquier aplicación profesional seria.

También otros fabricantes como Texas Instruments, Commodore, Atari, etc., siguieron sus pasos y construyeron sus propios ordenadores domésticos, cada vez más completos y más baratos.

Y mientras tanto... ¿qué hacía el líder absoluto?, se preguntará el lector avezado. ¿*Qué hacía IBM en este campo?* si es que hacía algo...

Aseguro que en IBM no tenían un pelo de tontos, así que ya en 1973 o 1974 se dieron cuenta de lo prometedor que para su cuenta de resultados podría ser esta nueva gama de ordenadores. Y tomaron la decisión de ir tomando posiciones en este tipo de tecnología. ¿Cómo hacerlo? Pues... evidente: ni más ni menos que como siempre había hecho las cosas IBM. ¿Es que acaso había otra...?

Así que, aplicando una vez más su *mantra empresarial*, es decir, vender sólo tecnología propia, comenzaron el diseño de un ordenador personal (bueno, no sé si exactamente *personal*, pero, al menos, *pequeño*): el **ordenador portátil IBM 5100**. Lanzado en 1975, fue una maravilla de diseño y empaquetamiento: dotado de un lector de cinta magnética en casete, teclado y pantalla, pesaba 25 Kilitos de nada y costaba sólo 9.000 dólares el modelo de 16 Kb y unos módicos 20.000 dólares el de 64 Kb. Podéis admirarlo en la ilustración siguiente (Wikipedia, Sandstein, CC by SA 3.0).



Para que os hagáis una idea, el Apple II, un par de años más tarde, tenía un precio de 1.298 dólares con 4Kb y 2.638 dólares si llegaba a 48Kb, como ya comenté hace un momento. O sea, el 5100 de IBM era... *un auténtico chollo*.

Efectivamente, fue una maravilla de diseño... **y un sonado fracaso comercial**. Aunque su venta continuó hasta 1981, nunca se vendieron muchas unidades de este IBM 5100.

Entonces, hacia 1980, el mercado de los ordenadores domésticos, pequeño todavía, estaba claramente dominado por Apple, con otros competidores como Commodore con su Commodore-64, competidores entre los que, en la práctica, no estaba IBM. Y las pocas máquinas que se vendían las compraban sobre todo aficionados e ingenieros con dinero abundante en su cuenta corriente para chapucear sus programillas en Basic o, fundamentalmente, para jugar a los primitivos juegos de ordenador de entonces, como el ping-pong y poco más. En realidad, se vendían pocas máquinas si lo miramos desde una perspectiva actual, pero eran desde luego muchas si lo comparamos con el resto del mercado de ordenadores en la época.



**Y a la sazón, de repente, en 1979 todo cambió.** Apple lanzó la **Hoja de Cálculo VisiCalc** como un producto más para su Apple II... y entonces, de golpe y porrazo, se materializaron de la nada millones de potenciales usuarios que antes ni sabían ni querían saber nada de estos “ordenadores personales”: Administradores, Contables, Financieros, Gestores, etc., hallaron una poderosa herramienta para ayudarles en sus labores *profesionales*. No para jugar, no: para hacer su trabajo cotidiano. Y esto sí que cambió todo el panorama.

ITEM	NO.	UNIT	COST
MUCK	43	12.95	556.85
BUZZ	15	6.75	101.25
TONER	250	49.95	12487.50
SNUFF	2	4.95	9.90
SUBTOTAL			13155.50
9.75% TAX			1282.66
<b>TOTAL</b>			<b>14438.16</b>

En la ilustración puede observarse cómo se veía la hoja de cálculo VisiCalc funcionando en un Apple II.

Gracias a VisiCalc, las ventas de Apple se dispararon. La propia VisiCalc fue al poco tiempo portada a otros ordenadores personales de la época y comenzaron también a aparecer otras hojas electrónicas clónicas, aprovechando el hecho de que a sus autores, Dan Bricklin y Bob Frankston, no se les ocurrió patentar su invento... y **nació la informática moderna**.

A IBM le pilló esta vez, cosa rara, a pie cambiado. Sus ingenieros y especialistas de marketing no habían anticipado este giro copernicano del mercado. Tenían, sí, un ordenador propio *pequeño* (no sé si llamarlo doméstico, personal, ni menos aún portátil) susceptible de ser ofertado, pero comercialmente muerto. Y rápidamente se dieron cuenta de la potencialidad que tenían estas nuevas máquinas para ofrecérselas a sus clientes de toda la vida: **las empresas**.

Pero... *¡no tenían ningún producto que ofrecer a sus clientes!*

Dos alternativas tenía IBM: Una, la instintiva, la que llevaban en los genes, es decir, seguir su práctica habitual y crear un nuevo sistema de tecnología propia poco menos que



de la nada. La otra, hacer de la necesidad virtud y apoyarse en tecnología ya existente de otros fabricantes para construir su propio ordenador personal. Con una decisión pragmática, visto el desastroso resultado comercial del 5100, optaron por la segunda, pues mandaron los plazos: como no podían permitirse un desarrollo de años hasta tener su producto listo, **optaron por la integración**. Un cambio radical en la estrategia tradicional de IBM.

Reunieron, pues, un grupo de ingenieros que, en menos de un año, habían seleccionado ciertos componentes prometedores del mercado, siempre que fuera posible de pequeñas compañías que no tendrían problema alguno en aliarse con el gigante, y los habían ensamblado para poder lanzar, a mediados de 1981, el IBM 5150, universalmente conocido como **IBM PC**.

Podéis observar en la imagen un genuino IBM PC de 1982, con sus dos disketeras y su monocroma pantalla (aunque está apagada, aseguro que es monocroma).



Entre estas pequeñas compañías estaba cierto pequeño fabricante de chips llamado Intel y también una pequeña compañía de Seattle conocida por su Basic y por haber creado, o por lo menos *estar en el proceso de crear* Xenix, una versión de UNIX de 16 bits: me refiero a una tal Microsoft, a quien encargaron la confección del Sistema Operativo para el IBM PC.

Hubiera podido IBM elegir en su lugar **CP/M**, de Digital Research, líder indiscutible del mercado de Sistemas Operativos para este mercado durante esos años, pero fueron reticentes al acuerdo (los de Digital Research, quiero decir), seguros como estaban por entonces de su superioridad tecnológica, mientras que Microsoft firmó todo (*casi* todo, en realidad) lo que IBM le puso por delante.

Y entonces Microsoft desarrolló (o compró, o adaptó, o de todo un poco... años más tarde, aún no está muy claro el asunto) PC DOS 1.0, lo que poco después se convertiría en MS/DOS.

Aún así, los primeros PC's de IBM ofrecían ambos Sistemas Operativos: PC DOS y CP/M, para que los usuarios eligieran, sólo que este último, que era obviamente superior, era también más caro... y eso le costó su supervivencia. Lo que de ningún modo ofrecía IBM era UNIX para su PC, o sea, el Xenix de Microsoft que Santa Cruz Operations portó finalmente a los procesadores Intel en 1983, dado que Microsoft no estaba muy dispuesta a apoyar nada que no fuera *su* DOS... e IBM tampoco. En una palabra, quien quisiera un SCO UNIX en su PC, debería de comprarlo aparte, instalárselo, mantenerlo, etc. Esto, en aquellos tiempos heroicos... fue simplemente demasiado para UNIX. Hasta bastantes años más tarde no comenzó este Sistema Operativo a tener una presencia algo más que

testimonial entre el parque de PC's... pero esa es otra historia y será contada en otro momento.

Esta historia del CP/M me recuerda a otra similar que ocurrió por la misma época con los diferentes estándares de vídeo doméstico.

Philips poseía una excelente tecnología propia, llamada “Video 2000”. Como era tan buena, con su gran resolución y su depurado sistema de imagen y Philips, inventora del casete de música, era una compañía líder en electrónica de consumo, se sintió suficientemente fuerte como para comercializar su tecnología en solitario. Por lo tanto, todo el que quisiera un “Video 2000” debería adquirir un reproductor de Philips y sólo Philips o de marcas *afines* (segundas marcas de ella, etc).

También Sony tenía una tecnología muy buena, quizá algo peor que la de Philips, de nombre “Beta”. Sony, el líder absoluto en electrónica de consumo, tomó exactamente la misma decisión que Philips... y a ver quién podía más. Pero el caso es que también había danzando por allí otra tecnología de vídeo doméstico de mucha peor calidad que las otras dos, de nombre “VHS”, creada por Matsushita, propietaria de JVC, que fue la compañía que lanzó inicialmente el sistema. Consciente de sus claras limitaciones técnicas, JVC decidió ceder los derechos de fabricación por dos perras a todo aquel que quisiera entrar en el incipiente mercado del vídeo doméstico...

...Y la historia es bien conocida: el VHS arrasó a su competencia, a Video 2000 y a Beta, que quedaron como curiosidades de museo. Aunque, algunos años después, también el propio VHS está prácticamente desaparecido... pero ésta es otra historia y será contada en otro momento.

*La moraleja: No siempre la mejor tecnología triunfa, sobre todo si no viene acompañada de un precio adecuado.* Se ve que IBM no aprendió en las peladas barbas ajenas, y esto a poco le cuesta la supervivencia diez años más tarde... pero también ésa es otra historia y será contada en otro momento.

Volvamos a nuestra historia. La que sí que estamos contando...

Estamos en 1981, e IBM acaba de lanzar su IBM PC: Procesador **Intel 8088** a 4,77 Mhz (*de un solo núcleo*, por si os lo estabais preguntando), 16 Kb de memoria (ampliable a 256 Kb), lector/grabador de cinta magnética o unidad de disco flexible de 5,25 pulgadas, monitor monocromo y mayormente PC/DOS como sistema operativo.

Como software de gestión, la inevitable hoja de cálculo, VisiCalc o cualquiera de sus clones, incluyendo desde 1983 a Lotus 1-2-3, que se impuso rápidamente como líder de este segmento hasta fines de los noventa. También había algún producto para escribir documentos, inicialmente el DisplayWriter (aunque antes que eso ofrecía IBM cosas aún más exóticas, como un producto llamado Script, por ejemplo).

De todo esto nosotros, en España, nos enteramos algún tiempo después, pues no fue hasta fines de 1982 que IBM comenzó a vender aquí el IBM PC. Supongo que parte del tiempo lo emplearon en traducir el PC/DOS y los manuales al español. Una vez abierta la brecha, los siguientes lanzamientos ya llegaron aquí con muy poco tiempo de demora sobre el lanzamiento en Estados Unidos.

En realidad, ateniéndose a sus características técnicas, este IBM PC no era gran cosa, incluso comparándolo con sus competidores de la época. Pero la maquinaria de marketing de IBM era formidable y en pocos meses había convertido al IBM PC en un gran éxito comercial.

Se vendió a particulares (jugadores, aficionados e ingenieros, los de siempre, vamos), aunque en este mercado Apple II y los demás mantuvieron sus ventas, pero sobre todo se vendió a Empresas, cuanto más grandes, mejor, allí donde la presencia comercial de IBM era ya muy importante: Contables, Administrativos, Gestores, Directores, Secretarías de Dirección, etc., comenzaron a ser usuarios de un PC.

Dos años después, IBM sacó al mercado el **IBM PC XT**, ya con disco duro de 10 Mb y un procesador Intel 8086, y, a fines de 1984, el **IBM PC AT**, equipado con un procesador Intel 80286, disco duro de 20 Mb, y pantalla de color CGA, y provisto de un flamante MS/DOS 3.0, mucho más fiable que las versiones anteriores.

Por cierto, este IBM AT *made in Ireland* (todavía salía a cuenta fabricar cosas en Europa), con sus 640 Kb y su MS/DOS 3.0 reluciente, fue el primer PC que tuve yo en mi mesa, para mí solito, a principios de 1986. Sus 20 Mb de disco duro... ¡no había manera de llenarlos, por mucho que metieras allí!

El Sistema Operativo venía en dos, creo recordar, floppies de 5,25 pulgadas y 360Kb cada uno; en uno, lo que era el sistema en sí, y en el otro, los programas adicionales, como el sort y tal. Aquello era como revivir la época del NCR Century 200, pero esta vez con 640Kb de memoria RAM, así que yo, literalmente, *nadaba en la abundancia*.

Para que os hagáis una idea, el Banco pagó cerca de seiscientos mil pesetas de la época a IBM por ese PC AT (unos 3.600 Euros). Por entonces mi sueldo, que era bastante bueno tras más de diez años trabajando en un trabajo bien remunerado como era la Informática, sería de alrededor de tres millones de pesetas (unos 18.000 Euros) anuales. Estas máquinas eran buenas, pero eran terriblemente caras. No obstante, las empresas las compraban...

...**E IBM nadaba en el éxito...**

...*Y sin embargo...*

...Y sin embargo, *no era suficiente*. Vale que eran los ordenadores personales más vendidos, pero las cifras de venta, jugosísimas, seguían sin satisfacer a IBM. Veamos las razones, recordando que estamos en 1984, año arriba, año abajo.

El mercado de PC's estaba dominado de largo por IBM. MS/DOS, con su propio nombre o como PC-DOS, estaba desplazando rápidamente al resto de Sistemas Operativos, y el esfuerzo publicitario de IBM estaba dando sus frutos. Pero éste seguía siendo un mercado relativamente pequeño, pues ni por prestaciones ni por precio era un cacharro que se vendiera bien entre la población en general, fuera de unos pocos *frikis* con dinero que quisieran jugar con ellos. Además, esta situación tardaría bastantes años en cambiar radicalmente, sobre todo con la bajada de precios y aumento de prestaciones y de software disponible de la segunda mitad de los 90, así como la popularización de internet por esas mismas fechas.

Lo que pasaba es que el mercado doméstico, el de los pequeños y baratos ordenadores dedicados fundamentalmente para jugar estaba dominado de largo por un excelente, pequeño y muy compacto micro-ordenador que fue bautizado como **Sinclair ZX-Spectrum** como el de la ilustración, que seguramente que trae buenos recuerdos a muchos de vosotros.



Ya sus predecesores, los ZX80 y ZX81 habían tenido una buena aceptación, pero fue el Spectrum el que de verdad revolucionó el mercado y consiguió unas espectaculares cifras de venta. Su diseño compacto, su gran capacidad (de 32 ó 64 Kb, de los que 16Kb eran la ROM donde tenía almacenado el Sistema Operativo y el Basic) y el hecho de usar baratas cintas standard de casete de audio como soporte, y todo ello por un precio más que razonable (treinta o cuarenta mil pesetas, unos doscientos euros), le convirtieron en un sonado éxito de ventas. ¡Tanto como para que a los pocos meses hubieran aparecido aquí y allá clónicos del Spectrum!

Y para los usuarios que sólo querían jugar y no querían lidiar con el Basic, ni pelearse con los casetes (éste era mi caso: yo nunca tuve un Spectrum, aunque la mayoría de mis amigos y compañeros sí que lo tuvieron), había otra alternativa: las consolas de juegos.



De todas las consolas de la época, la reina fue la **Atari 2600** como la de la imagen, que se conectaba al televisor mediante un cable coaxial, que traía los juegos en unos cartuchos ROM que se insertaban en una ranura y que permitían jugar a dos jugadores simultáneamente o a uno sólo jugando contra la propia consola.

Todavía debo tener en el cuarto trastero una consola Atari llena de polvo, eso sí, con sus juegos, además. Algunos de ellos eran muy impactantes para la época, como un ajedrez que jugaba bastante mal, pero bueno, unas damas a las que no había manera de ganar, al menos yo no lo conseguí nunca, o un backgammon muy divertido. Para esta consola estaba diseñado el juego de “E.T., el Extraterrestre” del que se han desenterrado no hace mucho millares de copias del desierto de Arizona. Yo fui uno de los pocos que lo compraron y no me extraña que acabaran enterrándolo: ciertamente era malísimo.

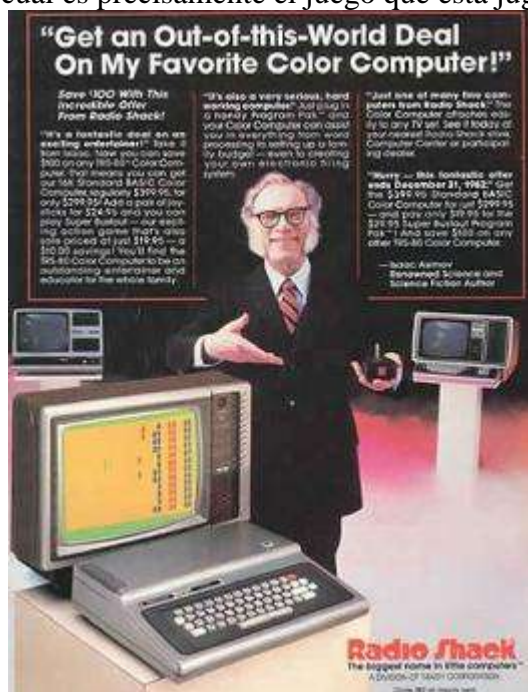
Además de la competencia del Spectrum, que IBM en realidad no veía aún como competencia de ellos (pues estaba dirigido a un mercado doméstico de bajas prestaciones y bajo precio que no le interesaba mucho... todavía), a IBM empezaban a llegar señales inquietantes que podrían amenazar su reinado en el mundo del PC profesional y *serio* (tradúzcase serio por *caro*): al ser las especificaciones del PC abiertas, cosa que contribuyó

fuertemente a su gran éxito, también era más fácilmente copiable.

### *Quid pro Quod.*

Columbia Data Products lanzó en 1982, tan sólo unos meses después del lanzamiento del IBM PC, su primer clónico; pocos meses después *una tal* Compaq lanzó al mercado el primer clónico “portátil” (bueno, más o menos portátil: más bien era *arrastrable*) del IBM PC, que fue seguido a los pocos meses o años por otras muchas compañías, incluyendo los grandes fabricantes como Digital, Hewlett-Packard, Philips, Olivetti, etc.

En la siguiente ilustración podemos admirar un anuncio de un ordenador doméstico ¡en color! ya en 1982: el **Radio Shack**, que no era en realidad un clónico del IBM PC, sino que se encuadraba en el mismo segmento de mercado que los Commodore 64: Basic, juegos, y poco más. Este ordenador se vendía por menos de 400 dólares, lo que ya era un precio muy ajustado para la época. Nótese que como pantalla utilizaba un televisor, y fijaos bien en cuál es precisamente el juego que está jugando... igual os suena de algo.



Debía ser, sin duda, una máquina maravillosa cuando fueron capaces de fichar nada menos que al indiscutido *Maestro* de la ciencia-ficción universal, Isaac Asimov, para que apareciera en su campaña publicitaria.

De calidad muy similar y menor precio, todos estos clónicos podrían amenazar la supremacía de IBM, que basaba su mensaje comercial en la gran calidad de sus máquinas, cosa que el mercado reconocía, lo que justificaba su mayor coste... sólo que esto último el mercado no lo tenía, ni mucho menos, tan claro.

En el mercado empresarial IBM siguió siendo líder muchos años, pero no así en el mercado doméstico, donde la competencia comenzó de nuevo a ser feroz, si es que había dejado alguna vez de serlo.

En el otro lado del espectro, el mercado de mainframes seguía dominado por IBM con muchos cuerpos de ventaja. Su posición era poco menos que de monopolio. Siguiendo

la tendencia marcada por los propios programadores, tal como conté en el capítulo correspondiente, los usuarios finales de los mainframes (administrativos, contables, gestores, etc.) tenían ya o tendrían muy pronto una abultada pantalla en su mesa conectada al ordenador central mediante un cable coaxial para cada una; podéis imaginaros el lío monumental de cables que había por aquellas épocas en las empresas.

Así podían realizar las operaciones de la compañía mediante las transacciones online de que disponían ya casi todos los Sistemas de Información. Muchas de estas pantallas eran también de IBM, las IBM 3270, pero había también otras pantallas clónicas de calidad y más baratas que le restaban muchas ventas.

Por fin, en el mercado de los minis IBM era un competidor más entre muchos, y su tarta de mercado no era especialmente jugosa. En cuanto al mercado de terminales especializados, sobre todo los bancarios, donde el nuevo IBM 4700 había sustituido al venerable 3600, IBM tenía también seria competencia, y aunque mantenía una cuota de mercado decente, no sólo no aumentaba, sino que incluso iba disminuyendo poco a poco.

Y entonces... **¿cómo aumentar las ventas, qué hacer para copar mercados donde IBM no era líder y asegurar aquellos donde sí lo era?** Es una pregunta poco original, ya lo sé, pero muy pertinente.

En este punto, *por exigencias del guión*, debo volver brevemente a **la situación de la Banca española a mediados de los ochenta**. En otros sectores probablemente pasaban cosas parecidas, pero me centraré en el Sector que conozco bien.

Los años ochenta vivieron una importante crisis bancaria en España. Muchas entidades pequeñas y medianas *fallecieron* y debieron ser fusionadas con otras, o adquiridas y puestas bajo la tutela de las de mayor tamaño, incluso de entidades extranjeras que aprovecharon la situación para desembarcar en España. Algún tiempo después, también las entidades más grandes tuvieron problemas, algunos de ellos muy serios, y se fueron produciendo fusiones entre ellas, dando origen a esos Bancos y Cajas de Ahorro tan conocidos hoy en día, en cuyo nombre oficial encontramos o encontrábamos una retahíla de nombres de extintos bancos o cajas absorbidos de grado o por fuerza a lo largo de los años; en una palabra, lo mismo que vivimos o hemos vivido recientemente... La historia se repite.

La causa de estas convulsiones (las de la década de los 80) fue, entre otras, la mayor apertura del mercado, obligada por la necesaria incorporación de España al Mercado Común Europeo, que en España se materializó en 1986 e hizo que la Banca española dejara de ser exclusivamente nacional y protegida por mil leyes, para competir en un mercado amplio, europeo en principio, luego global. Algunas entidades se adaptaron, pero otras no fueron capaces y quedaron por el camino.

Consecuencia directa de este nuevo entorno competitivo fue el tremendo aumento de la oferta bancaria a la población en general: muchísimos particulares de primeros de los ochenta no tenían cuenta bancaria, si es caso, una libreta de ahorro con unos miles de pesetillas (traducción para los jóvenes: algunos cientos de eurillos).

Este aumento de oferta fue tanto en productos ofrecidos como en puntos de venta, o sea, oficinas. Fueron los años de la enorme proliferación de oficinas bancarias por todo el territorio y fueron también los años donde se comenzó la comercialización masiva de nuevos productos, el primero de ellos, la propia Cuenta Corriente, y después otros muchos, como Tarjetas de Crédito o Débito, que casi nadie tenía en España antes de esos años,

Supercuentas, Fondos de Inversión, la generalización de la domiciliación de recibos, pues antes de esos años lo normal era que un amable cobrador se acercara a tu casa cada mes a cobrar el recibo de la luz o del teléfono, etc.

Pero además de esta creciente **apertura comercial**, también se produjo una importante **apertura tecnológica**: Todas las operaciones, casi sin excepción, deberían ser online; todos los empleados de las sucursales deberían tener acceso al sistema, y no sólo los cajeros, como hasta entonces; se empezaron a instalar cajeros automáticos literalmente en cada esquina, etc.

**La tecnología en sí se convirtió en un elemento de marketing más** para las entidades financieras.

En una palabra, la inversión en la tecnología necesaria para adaptarse a los nuevos tiempos comenzó a crecer fuertemente... y no precisamente en el entorno de los ordenadores centrales, que sí que crecía, pero no tanto, sino en el equipamiento de las oficinas. Y ordenadores centrales habría uno por entidad, quizá dos o tres, pero oficinas... de éstas las hay por cientos, o por miles.

Y es en este punto donde **mis dos historias más o menos paralelas convergen**: Porque IBM se encontró con dificultades en su mercado tradicional de tecnología, por la competencia de otros fabricantes que sabían bien lo que se hacían y porque su política de “bueno-y-caró” comenzó a tener problemas para ampliar ventas en un mercado bancario en expansión que estaba requiriendo fuertes inversiones en tecnología.

Básicamente, los responsables de Tecnologías de la Información de los Bancos y Cajas de Ahorro le decían a los comerciales de IBM: “Vuestra tecnología es la mejor, vale. Pero no puedo pagarla. Así que me conformaré con la de Fulanito, que, aunque es peor, sí que puedo pagarla. *No es personal, son negocios...*”.

Y entonces... **¿Cómo reaccionó IBM?**

Inventando una solución brillante... aunque también dolorosa (para ellos, quiero decir): **Sacrificó su querida tecnología propia especializada y convirtió a los IBM PC's en los terminales bancarios**. Y, más adelante, en casi cualquier terminal: de punto de venta, “pantalla tonta”, etc. Gracias a sus especificaciones abiertas, a su capacidad de conexión mediante las primitivas Redes Locales (Token Ring, en el caso de IBM; Ethernet vendría años más tarde) y a la facilidad de ampliación usando los slots libres, este trabajo pudo hacerse en un espacio corto de tiempo.

Para ello, IBM diseñó placas específicas que permitían a un PC emular el comportamiento de los terminales bancarios especializados, conectados a los dispositivos *extraños* que se necesitaban (impresoras bancarias, lectores de cheques, dispensadores de dinero, etc.) y con la misma conectividad con los sistemas centrales que los antiguos terminales especializados. Y, para garantizar la *protección de la inversión*, escribió también el software de emulación que permitiera correr el mismo software y las mismas aplicaciones que ya funcionaban en sus IBM 3600 ó 4700, o bien para construir desde cero el software de las oficinas, en el caso de nuevos clientes.

Con todo ello consiguió, fundamentalmente, **dar una respuesta moderna y eficaz a las necesidades de tecnología** de los Directores de TI y, sobre todo, **a un precio más que competitivo**.

Como por aquella época también empezaron a aparecer *tarjetas de emulación 3270* que eran capaces de hacer que el PC emulara a un terminal “tonto” del mainframe, también



se comenzaron a instalar PC's en lugar de pantallas *tontas* en los centros administrativos, pues además de su uso primario, que era acceder a las aplicaciones online del mainframe, también servirían como terminal ofimático, con las muy limitadas posibilidades de la época: hoja electrónica y procesador de textos, fundamentalmente, aunque esto también iba a cambiar radicalmente en unos pocos años.



La imagen anterior, también gentileza de nuestro buen amigo Jaume, muestra el Manual y los floppies (de 5,25", como podéis ver) del DisplayWriter 4, el producto estrella de IBM para la edición de textos a finales de los años ochenta.

Antes que eso, IBM tenía en el mercado un DisplayWriter 3, más primitivo todavía, con el que yo aprendí a usar un procesador de textos en PC, y antes aún tenía una *cosa* (no sé si llamarla “procesador de textos”) con funcionalidad copiada del mainframe, llamada *Script*, que ya cité antes y que, de parecerse a algo moderno, lo sería a escribir los documentos directamente en HTML.

**El éxito de IBM fue importante**, nuevamente. Comenzaron a proliferar PC's en oficinas bancarias, tiendas, comercios y otros establecimientos donde poco antes era impensable que estuvieran, así como en la mayoría de oficinas centrales, y muchísimos de ellos, de IBM, generando los años dorados (o sea, *de oro*) para el coloso del momento. Para que os hagáis una idea de los dorados que eran, a mediados de la década de los ochenta **IBM facturaba ella sola en equipamiento informático más que los siete siguientes competidores** (DEC, Univac, NCR, Burroughs, Hewlett-Packard, etc)... **juntos**.

*...Y fueron felices y comieron perdices...*

**...Pues no.**

La competencia tampoco se quedaba atrás, así que reaccionaron. *Vaya si reaccionaron...* y unos años más tarde pusieron en graves aprietos al antiguo dominador. Pero ésa es otra historia y será contada en otro momento... en concreto, unos pocos capítulos más adelante.

Efectivamente, los años ochenta del Siglo XX vivieron dos importantes convulsiones en la tecnología informática, que con el devenir del tiempo se han convertido en indiscutibles. Una fue, efectivamente, la irrupción en escena de los PC's que acabamos de destripar.

La otra fue la aparición en escena de las **Bases de Datos Relacionales**. Con ellas seguiremos nuestra historia.



## 10 - El camino hacia las Bases de Datos Relacionales

En efecto, junto a la de los PC's, la aparición de las Bases de Datos Relacionales a fines de los años ochenta es un hito clave sin el cual sería imposible comprender la informática actual. En este capítulo y en el siguiente intentaré contar cómo fue su introducción, al menos como yo la viví... Éste versa sobre todo lo que aconteció *antes* de que llegaran las propias Bases de Datos Relacionales, o sea, cómo andaba el patio tecnológico cuando se anunció la primera de ellas comercial, SQL/DS, la precursora de DB2. Será en el siguiente capítulo donde entraremos de verdad en el *tema relacional*.

Por lo tanto, en este capítulo **no voy a hablar de Bases de Datos Relacionales**, sino que me centraré en sus antecedentes: daremos un repaso a la historia de las Bases de Datos, mejor dicho, de los Sistemas de Gestión de Base de Datos que existían en el momento, sobre 1984-85. Es más, comenzaré incluso antes, para conocer cómo eran las técnicas de almacenamiento de la información antes de que existieran las Bases de Datos de cualquier tipo, o cuando sí que existían pero daba lo mismo, pues debido a sus limitaciones no se podían usar...

Los primeros ordenadores sólo eran capaces de procesar una clase de ficheros: los secuenciales. Es decir, un conjunto de registros de información guardados todos juntos, uno detrás de otro, que se leían o escribían de uno en uno, uno tras otro, hasta acabar todo el conjunto.

Los primeros ficheros *informáticos* fueron en tarjeta perforada: ya la máquina de Hollerith para el censo de 1890 funcionaba con ellas. Un fichero en tarjetas perforadas es un bloque de tarjetas con perforaciones que representan los caracteres que componen la información del registro. Los bloques se procesaban todos completos: se comenzaba a leer desde la primera tarjeta hasta alcanzar la última. Por tanto, si por algún motivo era necesario acceder sólo a un registro, se debía leer secuencialmente todo el bloque, desde el principio, hasta llegar a él.

En el capítulo dedicado al Cobol, y gracias la benevolencia de Jaume, os mostré cómo era un programa Cobol en tarjetas perforadas. Para que os hagáis una idea, un fichero en tarjetas perforadas es la misma cosa, sólo que las tarjetas tendrían todas ellas la misma estructura y en ellas habría grabados exclusivamente datos, no programas.

El soporte fue evolucionando. Primero, **cinta perforada**, heredada directamente del telégrafo, que es para las tarjetas lo mismo que los rollos de papiro en que los antiguos griegos escribían sus tratados para los códigos encuadernados en que los copistas medievales copiaron los escritos originales, los pocos que sobrevivieron, desde luego.

Después, a partir de principios de la década de 1950, fue la cinta magnética la que tomó el relevo... y esta vez para quedarse. Una cinta de 2.400 pies de longitud (unos 800 metros), la habitual a partir de los años sesenta, grabada a 1600 ppi (*phrames per inch*, o caracteres por pulgada) podía almacenar unos 40 Mb de información... cuando los carísimos discos de la época podían almacenar quizá tres o cuatro.

Ahora sigue habiendo cintas magnéticas, sólo que las actuales son capaces de almacenar Gigas y Gigas de información a un coste ridículo. La aparición de las cintas

magnéticas supuso también un cambio sustancial en la forma de diseñar las aplicaciones: ahora era posible mantener un fichero maestro, por ejemplo el de Cuentas Corrientes, acumular diariamente en otra cinta los movimientos del día y entonces, leyendo simultáneamente ambos ficheros (¡el omnipresente *padre-hijo*!), escribir en una tercera cinta el fichero maestro actualizado. Este proceso igual podría hacerse en ficha perforada, nada lo impedía... pero es que, además de lento, ¡era carísimo! Las tarjetas perforadas no son reutilizables, pues una vez perforadas, no se pueden *des-perforar*, mientras que las cintas sí que lo son: se pueden leer y escribir una y otra vez sin merma en su funcionamiento, al menos durante mucho tiempo.

Pero esta situación no duró mucho: a finales de los cincuenta del siglo pasado, con la aparición de los discos magnéticos, la situación comenzó a cambiar. Con un disco magnético es posible algo que con una cinta no lo es: **acceder a un registro determinado de un fichero sin necesidad de leer previamente todos los registros anteriores**, simplemente dando instrucciones a la cabeza lectora de qué área concreta del disco leer. Y lo mismo con la grabación. Esta característica de “acceso directo” abría obviamente nuevas posibilidades que al poco se comenzaron a explotar. Aquí podéis admirar un anuncio publicitario de discos magnéticos de 1977.

Now you can get our disk systems  
within 30 days ARO at the  
industry's lowest prices:

- 80 Mbytes for under \$12K\*
- 300 Mbytes for under \$20K\*

Today's most demanding, most advanced system...  
The price listed above is the complete disk...  
Each system includes one high performance...  
...an optional...  
...the software of your choice.

When you buy disk systems from us, you'll know...  
...the same from our company on the ground floor...  
...you'll want to know more. Requesting more...  
...with delivery and consulting with our...  
...and customer support, we'll get you...  
...up quickly and keep you up to date...  
...our pricing information and technical details...  
...contact our System Industries representative in...  
...your area.

**System Industries**  
An equal opportunity employer  
10000 Wilshire Blvd.  
Beverly Hills, California 90210  
JFK 100-1000, 100-10000

• 800 prices 80-99 systems

Hay que fijarse tanto en sus capacidades como en su módico precio: 12.000 dólares USA de nada por nada menos que 80 Mb de espacio, o sea, 150 dólares por cada Mb, teniendo en cuenta que ¡se trata de dólares de 1977!

Para que un programa supiera en qué dirección física del disco está la información demandada (supongamos que la cuenta número 1537), **antes había que haber anotado, en algún sitio, en el momento de la escritura, la dirección física real donde había ido a parar dicha cuenta.**

Una solución obvia es utilizar el *propio número de cuenta como dirección* a la hora de grabar. Así, la cuenta 1537 estaría en la dirección 1537, y localizarla es inmediato. Pero, claro, quizá la numeración de las cuentas no permita esta solución, por ejemplo porque el propio número incluya el tipo de la cuenta, o porque tengan muchos huecos de numeración entre las cuentas, por ejemplo porque las cuentas adyacentes a la 1537 fueran la 1459 y la 1703, con lo que se desaprovecharía muchísimo espacio. En la práctica casi nunca puede

usarse tal tipo de direccionamiento, salvo para algunos ficheros-tabla de códigos, siempre que tengan relativamente pocos registros, sin apenas espacios entre ellos y que sean además muy estables en el tiempo.

Una forma de solucionar esto sería con la aplicación de una fórmula matemática que convirtiese el código del registro a buscar a una dirección física y unívoca. Esta técnica implica usar una **función hash**, y es muy eficiente... en los casos donde es factible usarla, que tampoco son tantos, y lo digo por experiencia.

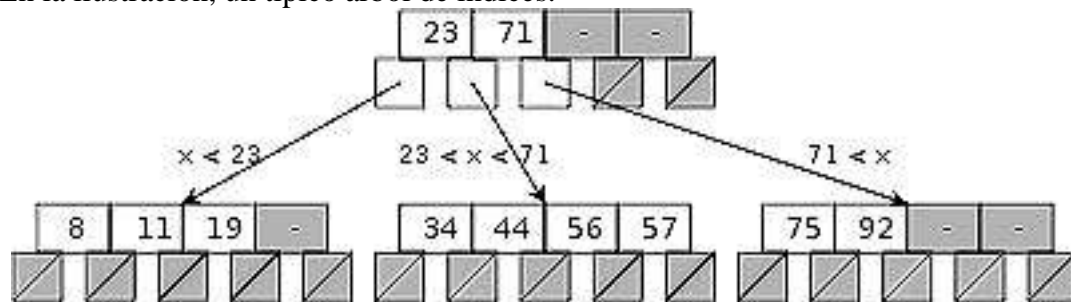
Así que rápidamente se constató que la única forma eficaz de poder conocer la dirección física de un determinado registro de un fichero era mediante la utilización de **índices**.

Un índice consiste ni más ni menos que crear, simultáneamente a la escritura del fichero que se pretende indexar, otro fichero diferente al principal que contendrá, ordenadas, las claves de acceso junto con sus direcciones. Así, tendríamos que nuestras cuentas estarían representadas, por ejemplo, con registros que podrían decir: 1459-007; 1537-008; 1703-009... Es decir, la cuenta 1459 está en el bloque 007, la 1537 en el 008...

Ahora, para acceder a la cuenta 1537 basta con leer el fichero de índices, mucho más corto que el principal, hasta determinar la dirección de la cuenta y entonces ir al fichero principal, mediante un acceso directo, a dicha dirección. Es más, si es posible mantendremos en memoria principal el propio fichero de índices, agilizando muchísimo el acceso, pues así evitamos leer cada vez el índice en el disco.

Pero los propios índices pueden resultar de buen tamaño, así que se crea a su vez un índice para el propio índice, y así sucesivamente hasta llegar a un tamaño de índice razonable para mantener en memoria, creando así un árbol de índices. En realidad las cosas son *un poco más complicadas* de lo que he explicado aquí, pero creo que para hacerse una idea es suficiente... y los que ya conocéis todo esto... pues eso: ya lo conocéis.

En la ilustración, un típico árbol de índices.



Ya desde el advenimiento del Cobol en 1960 se incluyó la definición y tratamiento de ficheros indexados mediante la cláusula ORGANIZATION IS INDEXED y usando diversas cláusulas de definición y extensiones a las sentencias de lectura y escritura. Esto permitió diseñar las primeras aplicaciones con acceso directo a la información. En inglés se llama acceso *Random*, es decir, aleatorio, pero en realidad no es aleatorio en absoluto, sino que tu programa decide a qué registro acceder en función de la información solicitada. Las aplicaciones que típicamente necesitan acceder a información de esta forma son las **Aplicaciones Online**. Es decir, si fue posible inventar el Online fue ni más ni menos que gracias a la existencia de los índices...

...Porque lo interesante de todo esto es que **el acceso mediante índices ha sido, y sigue siendo, el método utilizado por todas las Bases de Datos de todo tipo para acceder a la información.**

Utilizando **ficheros secuenciales indexados** se pudo empezar a desarrollar las incipientes aplicaciones online, aunque hasta que no se dispuso de Gestores de Teleproceso era realmente difícil programar estos sistemas que, además, tenían muy poca capacidad: a los exiguos tamaños de memoria y procesador existentes, se añadía la escasa velocidad de las carísimas líneas telefónicas de entonces: punto a punto y de 1200 o 2400 *baudios* (bits por segundo).

**Una curiosidad:** uno de los primeros Monitores “serios” de Teleproceso (anterior en varios años al ínclito CICS de IBM), fue el **PCL**.

El acrónimo PCL significa **Programa de Control de Líneas** (*en español, sí*), dado que fue desarrollado en el laboratorio de IBM en Barcelona por un equipo dirigido por el holandés Rainer Berk, bajo las especificaciones del cliente y trabajando codo a codo con ellos, que se instaló por primera vez en La Caixa d’Estalvis i Pensions (tras varios cambios de denominaciones, en el momento de escribir este capítulo se llama CaixaBank), alrededor de nada menos que 1964. Este programa, que fue evolucionando y mejorando con el paso de los años, se usó durante los primeros tiempos en todos los bancos españoles que empezaron a hacer pinitos con su teleproceso en la década de los sesenta y primeros setenta, pues hasta al menos 1970 ó 71 no comenzó IBM a ofrecer CICS en España, e IMS/DC es posterior en cinco años o más.

No esperéis que cite bibliografía, o algún artículo de la Wikipedia o de donde sea hablando de PCL, ni ningún otro sitio hablando de este Programa de Control de Líneas... ¡**No hay!** Es como si *nunca hubiera existido*. Fue un hito español en informática... y prácticamente nadie lo conoce, ni se encuentra documentación, aparte de en la memoria de algunos viejos rockeros.

Y, aunque no tanto, prácticamente lo mismo sucede, por ejemplo, con otra gran innovación española de la época y, además, ésta sí fue completamente española: la **Red Especial de Transmisión de Datos** (RETD), que fue la primera red mundial de transmisión de paquetes, sobre la que también ha caído el olvido, aunque afortunadamente Jesús Martín Tardío (ingeniero de Telefónica de aquellos tiempos heroicos) ha escrito un maravilloso relato de lo que pasó durante esos años gloriosos que, si con suerte aún existe el fichero, podéis encontrar aquí: <http://personales.ya.com/tardio/Zips/Td.pdf>.

Ignoro si será exclusivamente tradición española la de ignorar (o peor, ¡despreciar!) las cosas buenas que se han hecho y magnificar las malas, pero indudablemente eso ocurre, y hay muchísimos ejemplos. **Uno flagrante**, que no tiene nada que ver con ordenadores:

¿Sabéis quién fue el Almirante Nelson? ¿Sabéis quizá qué pasó en la batalla de Trafalgar en 1805? Estoy seguro de que ambos os suenan... bastante. Incluso se celebró el bicentenario hace pocos años, con asistencia de autoridades, discursos, desfiles y todo.

Sigamos. ¿Sabéis quién fue el Almirante Blas de Lezo, alias *Patapalo*, alias *Medio-Hombre*? ¿Sabéis por ventura qué ocurrió en el Sitio de Cartagena de Indias sesenta años antes de Trafalgar, en 1741? Pues resulta que Medio-Hombre, con seis naves, 3.000 hombres y 600 arqueros indios derrotó a la fuerza sitiadora, una colosal flota de 186 buques al mando del Almirante inglés Edward Vernon, la mayor reunida hasta la fecha, con cerca de 30.000 hombres, en el transcurso de la llamada “*Guerra de la Oreja de Jenkins*”.

Trataba Edward Vernon de ganar Cartagena de Indias, la llave para que Inglaterra se hiciera un hueco en las Américas, y se encontró con el Almirante Blas de Lezo que, con unas fuerzas infinitamente inferiores, infligió a la Armada británica su mayor derrota de la Historia. La consecuencia fue asegurar el dominio absoluto español sobre las aguas

americanas hasta bien entrado el Siglo XIX... cuando la América española dejó de ser española para ser americana. Si se hubiera perdido Cartagena de Indias, igual ahora la América española sería inglesa, quién sabe con qué consecuencias para todo el mundo.

Lo curioso del asunto es que en España poquísima gente conoce la historia de *Patapalo* u otras similares que han acontecido a lo largo de la Historia, mientras que todo el mundo conoce la paliza que Nelson dio a la Armada franco-española en Trafalgar, o el desastre de la *Armada Invencible*, desastre que, por otra parte, tampoco fue tanto desastre como nos han enseñado... ¿por qué será?

En Inglaterra, por ejemplo, ocurre exactamente lo mismo: también todo el mundo sabe de la Invencible y de Trafalgar, que tiene incluso dedicada una céntrica y muy turística plaza en Londres con majestuosa estatua en el centro y todo, pero casi nadie ha oído nada acerca del Sitio de Cartagena de Indias...

En fin; perdonad esta diatriba de viejo cascarrabias. Dejemos a un lado la Historia Universal y volvamos con los ficheros indexados...

Efectivamente, la capacidad de los ficheros indexados de recuperar o grabar la información mediante un acceso directo, sin necesidad de leer el fichero completo, permitió el advenimiento de las primeras aplicaciones online... pero si las aplicaciones en sí eran técnicamente posibles, en realidad existían otras muchas dificultades para su normal explotación diaria.

Una de ellas era la dificultad para organizar los accesos concurrentes, es decir, la gestión de los bloqueos. En un proceso bancario, por ejemplo un pago de cheque, es normal acceder primero a la cuenta para ver, lo primero, si existe, y después, si tiene saldo suficiente para hacer frente al cheque, luego a los talonarios de la cuenta para comprobar que el cheque existe y no está bloqueado por algún motivo, etc. Al final de todas las comprobaciones se marca el cheque como pagado, se actualiza el saldo y se graba el movimiento para su contabilización.

Si todo va bien, no pasa nada. Pero puede ocurrir que otra transacción que se ejecuta en la región de al lado, que para algo habíamos inventado ya la multiprogramación, pretenda pagar un recibo de la misma cuenta en el mismo momento, usando, además, los mismos dineros que van a usarse para pagar el cheque. Si no se controla muy bien y se obliga a la segunda transacción a esperar a que la primera termine, se puede montar un lío de mucha consideración. Con los ficheros indexados todo este montaje hay que controlarlo a mano, complicando enormemente la programación... en unos ordenadores que, recordad, sólo tienen unas pocas KB de memoria.

Pero la otra es aún más seria: *Ante un error de hardware o de programa* (que, a veces *cascan*, ¿sabéis?, por muy bien escritos que estén) *el fichero indexado se queda hecho unos zorros*. Como en realidad se mantienen dos ficheros a la vez, el de índices y el de datos, es muy posible, es más, era lo normal, que el fallo deje actualizado uno de ellos, pero no el otro... dejando el fichero inservible. Entonces, la única alternativa era asumir que el contenido del fichero principal, o sea, los datos, es el correcto lo sea o no lo sea, pues no hay otra posibilidad, y a partir de él reconstruir los índices, utilizando el fichero en exclusiva durante el tiempo que dure el “rebuild”... y rezar porque todo acabe bien y no haya que volver a la versión anterior, perdiendo toda la sesión online hasta el fallo. O sea, parando la aplicación online mientras se recupera... y sus usuarios, mientras tanto, mirando. Bastante inaceptable, como podéis suponer.

Fue otro de los gurús de la informática, **Charles Bachman**, quien estuvo en la génesis de la solución a todos estos problemas, acuñando el concepto de “**Base de Datos**”. De sus trabajos nació la primera Base de Datos de la historia: **IDS**, de General Electric. GE era una de las grandes compañías mundiales y lo era también en informática; de hecho durante los años sesenta era uno de los así llamados “*Siete Enanitos*”, las siguientes siete mayores compañías de informática que competían en el mercado con la *madrastra*, IBM, hasta que decidió abandonar el negocio tan pronto como en 1970, vendiendo la división de informática a otro de los *enanitos*: Honeywell.

La idea fundacional de las Bases de Datos (en realidad, de los *Sistemas Gestores de Bases de Datos*, o *DBMS*) era solventar todos los problemas derivados del uso aislado de los diferentes ficheros, no tanto pensando en el batch, que, en realidad, estaba resuelto sin necesidad de Base de Datos alguna, sino para dar soporte a los emergentes procesos online y de tiempo compartido.

Un DBMS debía, en primer lugar, asegurar la **coherencia de los datos en todo momento**. Ésta es una *conditio sine qua non* para poder desarrollar y explotar con éxito una aplicación online. Y además debe permitir interactuar con ella a desarrolladores, técnicos de sistemas, etc., con *cierta* sencillez, para facilitar el desarrollo, explotación y posterior mantenimiento de las aplicaciones. Resumiendo, un DBMS de cualquier tipo que se precie debería, al menos, cumplir las características siguientes:

**1 Ante el fallo de un programa, debe ser capaz de recuperar la información** que éste se encontró, deshaciendo todos los cambios realizados antes de su fallo. Esto implica la existencia de un *log de cambios* donde quedan reflejadas todas las modificaciones efectuadas por el programa que además permita deshacerlas en caso de fallo para volver a la situación original.

**2 Debe ser, además, capaz de recuperar un proceso completo que haya resultado erróneo**, aunque haya terminado aparentemente bien, o sea, que no ha cascado. Por ejemplo, puede que un determinado proceso batch haya funcionado mal y actualizado erróneamente muchos registros, o todos ellos; el Gestor debe permitir recuperar el proceso completo para repetirlo una vez arreglado el programa fallón. Esto exige una gestión avanzada de copias de seguridad coordinada con la propia gestión del log.

**3 Debe proporcionar un procedimiento robusto ante fallos del propio sistema.** En el caso de un error en una Base de Datos, debe permitir recuperar la que esté afectada sin necesidad de afectar el resto; de esta manera se independizan unas aplicaciones de otras.

**4 Debe gestionar de forma eficaz la concurrencia entre procesos.** Esto exige un control automático y eficaz de los bloqueos: cuando un programa accede a un registro de la Base de Datos con la intención de actualizarlo, el DBMS debe dejarlo bloqueado para cualquier otro proceso hasta su terminación correcta, en cuyo caso los cambios se consolidan, o errónea, en cuyo caso se deshacen.

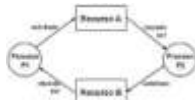


En la ilustración anterior se representan algunos de los métodos posibles para el tratamiento de los bloqueos, incluyendo el muy español del Avestruz: no hacer nada y luego... ¡ echar la culpa a otro!

Para lograr todo esto se requiere que el lenguaje de interfaz con la Base de Datos permita *avisarla* de que el programa tiene intención de actualizar el registro que está solicitando para lectura; que yo sepa toda Base de Datos tiene esta función, como las peticiones tipo “GHU” en IMS DB, o la “SELECT FOR UPDATE” en SQL.

Cuando se produce concurrencia, la estrategia correcta no es precisamente la del “*avestruz*”, ya digo, sino dejar en espera al segundo peticionario que llegó, hasta que el registro bloqueado quede liberado y entonces permitirle continuar su proceso. Pero esto también tiene sus problemas...

**5** ...Porque en el caso de que dos transacciones distintas requieran registros cruzados, lo que en la literatura técnica se conoce como “deadlock”, o, en *cheli*, el *abrazo del oso*, o se resuelve de alguna forma o quedarán bloqueados ambos registros... para siempre.



En la imagen podemos observar un ejemplo: Un proceso P1 requiere el registro B, y lo bloquea, y después el registro A, mientras que otro proceso P2 requiere el registro A, y lo bloquea, y después el registro B... y estas cosas se dan con más frecuencia de lo que podría parecer.

Si hay la mala suerte de que ambos procesos, ejecutándose simultáneamente, han obtenido sin problemas sus primeros registros... tenemos un problema. Porque el proceso P1 tiene bloqueado el registro B y está a la espera del A... que está bloqueado por el proceso P2, que a su vez está esperando el registro B, y así sucesivamente. Si el DBMS no reconoce el problema para ponerle solución, típicamente abortar uno cualquiera de los dos procesos para dejar que uno al menos de los dos, el otro, termine bien, y después relanzar el proceso abortado, podría colapsarse y requerir intervención manual, lo que en general no parece muy buena idea.

**El Gestor de Bases de Datos debe solventar los deadlocks cuando ocurren.**

Además, este caso puede darse no sólo con dos procesos, sino con cadenas de tres, cuatro... lo que es, ya os imagináis, bastante complicado de resolver.

**6** Las bases de datos **deben proveer una interfaz** para que los programadores puedan codificar correctamente los accesos, en lectura o actualización, o la navegación entre diferentes registros relacionados; interfaz documentada y única y a ser posible que libere al programador de todas las funciones de control, gestión de los índices, de los punteros, los backups o recoveries, etc. Deben proveer también toda una serie de programas de utilidad para todas las tareas técnicas, como reorganización, creación o recuperación de índices, etc.

**7** Y además... deben permitir al usuario (el usuario de estas Bases de Datos era siempre el programador, no el usuario final) la **abstracción del modelo físico**: es decir, se diseña el modelo lógico de la información, cómo son las entidades de datos y cómo se relacionan entre sí, y se plasma en el propio diseño de la Base de Datos. Naturalmente, esto

cambió, con el paso del tiempo, la propia forma de diseñar las aplicaciones, dando origen al poco tiempo al nacimiento de las metodologías de Diseño Estructurado, el modelo Entidad-Relación, etc. Pero ésta es otra historia y será contada en otro momento.

Todas estas características están más o menos resumidas en lo que se conoce como **ACID**: *Atomicidad, Consistencia, Aislamiento y Durabilidad*. En inglés, claro.

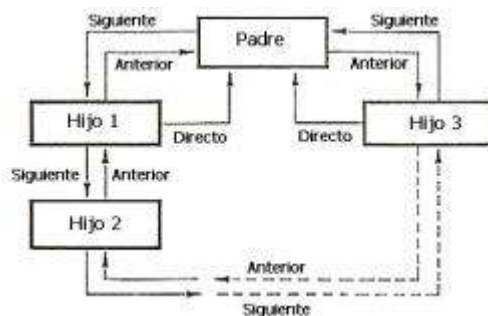
Podemos decir que **las Bases de Datos cumplieron sobradamente sus objetivos** desde el primer momento. Doy fe.

Evidentemente, las primeras de ellas no eran sencillas ni de diseñar ni de programar correctamente (bueno, si nos ponemos quisquillosos, *tampoco ahora*), pero efectivamente eran razonablemente seguras, rápidas y eficaces, cuando se rompían se arreglaban bien y rápido y, aunque complicadas de programar, lo eran mucho menos que si hubiera habido que hacerlo directamente con los ficheros indexados.

Programar para IMS era bastante complicado. Lo sé: yo lo hice. Lidar con las PSBs, las PCBs, las SSAs, etc., necesita de mucha, pero mucha atención por parte del programador... y conocimientos, claro, y tener siempre presente el diagrama del diseño físico de la Base de Datos.

Aún recuerdo una transacción que tenía un tiempo de respuesta normal (pequeño) y ante una aparentemente mínima modificación de pronto se convirtió en la transacción que más recursos consumía de todo el Banco. Resulta que la pequeña modificación obligaba al IMS a recorrer toda una larguísima cadena de gemelos (las distintas ocurrencias de un mismo segmento) para insertar el suyo... y nadie nos dimos cuenta de ese hecho: ni el Analista –o sea, yo, y mira que sabía yo de IMS por entonces–, ni el programador, que sabía también lo suyo, ni tampoco el Técnico de Sistemas...

Para solucionarlo se añadió en el diseño de la Base de Datos un nuevo puntero en el segmento padre para que apuntara a la última ocurrencia del segmento hijo de marras, el puntero PCL, por “Physical Child Last”... y el programa ahora solicitaba directamente ese último segmento antes de insertar el suyo. Un éxito: la transacción volvió a ser una de las más modositas de la instalación. Esta anécdota tontorrón puede ayudar a haceros una idea de lo fino que había que hilar.



A raíz de los esfuerzos de Bachman se creó en **CODASYL** un grupo para definir las características y especificaciones que debían cumplir las Bases de Datos en Red como la de la imagen anterior; estas especificaciones se publicaron en 1969 y sentaron las bases que permitieron el avance de la industria de las Bases de Datos.

Algunos, como IBM, habían avanzado por su cuenta y ya tenían en el mercado la suya propia (**IMS**) desde 1968, ya que fue diseñada para poder gestionar el complicadísimo



control de inventario del Programa Apolo, ése que hizo que Armstrong, Aldrin y compañía pisaran la Luna... aunque todavía hay quien lo duda.

Comenzaron a aparecer en el mercado diferentes bases de datos que implementaron, unas más y otras menos, los estándares CODASYL: unas fueron concebidas como bases de datos en red y otras fueron diseñadas como jerárquicas; el modelo jerárquico, que admite sólo relaciones de padre a hijo y entre gemelos, no es más que un modelo en red restringido, y por tanto estaba también contemplado en las especificaciones CODASYL.

En <http://infolab.stanford.edu/pub/gio/dbd/acm/app2.pdf>, tenéis una recopilación de las Bases de Datos de todo tipo que se habían creado y/o comercializado... antes de 1986, antes de la *explosión relacional*. Hay *unas pocas*: ¡Más de 150!, y aunque en puridad algunas de ellas no son en realidad Bases de Datos, sino más bien sistemas de acceso a ficheros, o aplicaciones escritas sobre una determinada Base de Datos, os podéis hacer una idea del gran follón “*basedatístico*” que había esos años.

Casi cada fabricante importante construyó y vendió la suya, y además aparecieron bastantes empresas independientes de software que construyeron y vendieron sus propias bases de datos. Probablemente éstas debieron ser de las primeras empresas que construyeron software básico sin ser también fabricantes de hardware.

A la primera de todas, **IDS** (de GE, luego Honeywell), le siguieron (además de **IMS**, de IBM), **DMS1100** (de Sperry Univac), **DMS II** (Burroughs), **IDS2** (Bull), **IDMS** (Cullinet, luego Computer Associates), **Total** (Cincom), **Datacom** (ADR, luego Computer Associates), **Adabas** (Software AG), **System 2000** (De una tal *INTEL* -antes *MRI*-, que no es la misma Intel que todo el mundo conoce; yo juraría que en España la vendía Siemens, para sus ordenadores Siemens 4004, aunque no lo puedo asegurar), etc.

Las tres últimas se basan en el concepto de “listas invertidas”, que proporciona un excelente tiempo de acceso en lectura, aunque no precisamente en actualización, lo que las hace especialmente eficientes en entornos de sólo-consulta donde haya poca o nula actualización.

Adabas, en concreto, sigue siendo utilizada en grandes instalaciones españolas y de todo el mundo después de su “*lavado de cara*” relacional de hace ya bastantes años, pues admite SQL como interfaz, aunque buena parte de la culpa de que siga estando vigente la tiene *Natural*, también de Software AG, el único lenguaje de Cuarta Generación que de verdad tuvo éxito, que funciona muy bien con Adabas (y con DB2, y con Oracle...) y que todavía se mantiene funcionando en la actualidad... pero esa es otra historia y será contada en otro momento.

Esto de las listas invertidas puede parecer antiguo, pero sigue siendo muy usado en la actualidad: por ejemplo, los buscadores que se usan en aplicaciones web, incluyendo el todopoderoso Google, utilizan variaciones de listas invertidas para realizar las búsquedas: necesitan muy buen tiempo de acceso en lectura, pero no tanto en actualización.

Desde luego, los nombres de la mayoría de Bases de Datos de la época parecían una *sopa de letras*. Tanto que, en aquellos tiempos gloriosos, no podíamos evitar, entre los enteradillos de la profesión, contarnos el siguiente chiste (que entonces tenía más gracia que ahora, cierto):

**En una tienda:**

**Cliente** (mientras señala): *Déme ése. Y déme ése. Y de ése... y de ése, dos.*

**Tendero** (al dependiente): *Déle uno. Total, debe dos.*

...Ya, ya avisé que no tenía gracia... Algo mejora si usamos los acrónimos, aunque no demasiado:

**Cliente:** DMS. IDMS. IDS...IDS2.

**Tendero:** DL/I. Total, DB2.

En fin. Al dichoso *Adabas* no se nos ocurría forma alguna de meterle en el chascarrillo...

La propia sobreabundancia de versiones de Bases de Datos de todos tipos y colores a finales de los setenta ya quiere decir mucho: tuvieron gran éxito comercial y cumplieron con las expectativas más halagüeñas.

Efectivamente resolvieron aquellos problemas que vinieron a resolver: las aplicaciones fueron mucho más fiables, los datos mucho más coherentes, el proceso de diseño de los datos más estructurado y lógico y, ante un fallo de cualquier tipo, la información se recuperaba normalmente de forma rápida y completa, sin pérdida alguna. Pero tuvieron también inconvenientes, algunos de ellos bastante importantes, que abonaron el cambio de tecnología de mediados y finales de los ochenta.

En primer lugar, **cada Base de Datos era de su padre y de su madre**. Literalmente. Cada una de ellas tenía un lenguaje de definición, un modo de ejecución, diferentes programas de utilidad y, por supuesto, diferente interfaz para los programas de aplicación. La forma usual era resolver los accesos mediante la llamada vía CALL, generalmente estática, a un determinado módulo del gestor (en el caso del IMS de IBM, por ejemplo, el módulo es CBLTDLI desde Cobol, ASMTDLI desde Assembler, etc.), con ciertos parámetros que le indican a qué Base de Datos se quiere acceder, a qué segmento o entidad, y para hacer qué: leer el segmento de cierta clave, o el siguiente en secuencia, modificar el segmento con nuevo contenido, borrarlo...

Hay que tener en cuenta que todas las Bases de Datos de la época, sin excepción, eran *Navegacionales*, es decir, era el programador el que indicaba cuidadosamente el orden de recuperación exacto de la información en la Base de Datos: primero recuperar un segmento padre con una clave dada, luego recuperar la primera ocurrencia de uno de sus segmentos hijo, leer en secuencia todos estos segmentos hasta cierta condición, reescribir el segmento, insertar un nuevo segmento al final de la cadena, saltar a otro lugar distinto, y así hasta acabar el proceso, tanto si es batch como si es online. Y en cada Sistema de Base de Datos este proceso era distinto, pero no *un poco* distinto, sino **completamente diferente**.

Migrar de una a otra Base de Datos requería no sólo efectuar un proceso complicado de descarga de la información y carga en la nueva Base de Datos, que en realidad era lo más sencillo, sino que había que reprogramar completamente todos los accesos... para lo cual había, en primer lugar, que dar un completo Plan de Formación a todo el personal, y luego reprogramar la Aplicación completa.

En una palabra: **No existía portabilidad entre Sistemas**. Ni la más mínima. Quizá esto no fuera un problema muy serio para las instalaciones de cliente, pues tampoco vas a andar cambiando de Sistema a cada rato, pero esa falta de portabilidad sí que era un inconveniente importantísimo para los fabricantes de Software de Gestión.

Veámoslo con un ejemplo: Supongamos que hemos desarrollado una Aplicación que queremos vender a muchos clientes, por ejemplo una Nómina. Inicialmente la desarrollamos en, digamos, un mainframe IBM con IMS, que para algo tiene la mayor cuota de mercado. Cuando está lista, la empezamos a vender... bueno, *antes* de que esté

lista, que el marketing es el marketing.

El caso es que, aunque haya una base importante de clientes para nuestra Nómina, resulta que no todos los clientes tienen esa configuración. Los hay que tienen un mainframe de IBM, pero la Base de Datos es Total, o Datacom, por ejemplo. Otros grandes clientes tienen un Siemens 4004 con System 2000, o con Adabas. Y otros, un Bull con IDS2. Migrar la aplicación para que corra simultáneamente, con las mismas funcionalidades, en todos estos sistemas y Bases de Datos tan diferentes entre sí es... una locura.

Migrar una aplicación de Unix a Windows o viceversa es un juego de niños comparado con lo que era en aquellos tiempos migrar de IBM/IMS a Digital/Adabas... ¡Y eso que, al menos, el Cobol sí que era el mismo! De hecho, la fragmentación de la tecnología, o, más exactamente, la diversidad de diferentes interfaces con los Sistemas, redujo muchísimo el alcance del incipiente mercado del software independiente durante las décadas de 1960, 70 y casi todos los ochenta del siglo pasado. Cosa que a los fabricantes de hardware no les preocupaba en absoluto, como es obvio: así, su mercado era cautivo. *A los fabricantes de cualquier cosa les encantan los mercados cautivos...*

Y, además de esta falta de estándares, es que **tampoco era nada sencillo diseñar y programar bien para ninguno de estos Sistemas y Bases de Datos.**

En igualdad de condiciones, era ciertamente mucho más sencillo que hacer lo mismo *a pedal* directamente con ficheros indexados, desde luego que sí, pero seguía siendo difícil... mayormente porque la mera existencia de las Bases de Datos permitió la realización de Aplicaciones con una complejidad muy elevada, que hubieran sido completamente imposibles sin esas Bases de Datos.

Se requería una muy buena formación, apoyo constante en manuales, pues había tal cantidad de opciones y posibilidades que necesitabas consultar los formatos de las llamadas al Gestor de Base de Datos constantemente... y experiencia. Mucha experiencia. Y lo primero se solventaba con cursos de formación, por caros que fueran, que lo eran, lo segundo con una buena fotocopia del original para cada uno... pero la experiencia sólo se conseguía con tiempo y aprendiendo de las galletas, sobre todo *de las propias*, que ya sabemos que nadie escarmienta en cabeza ajena.

En una palabra, **los buenos técnicos** (analistas, programadores, técnicos de sistemas, etc.) **escaseaban**. Escaseaban mucho. Se los reconocía porque, cuando hablaban, nadie en absoluto a su alrededor que no fuera de la profesión entendía una palabra. Y a veces, ni así. Por ejemplo:

*“Hemos tenido un OC4 en una región del VSAM, y el PTF del CAS no pudo instalarse correctamente porque la versión del HSM no estaba a nivel...”*

Ya me diréis lo que entendía nadie de todo eso y, sobre todo, ya me diréis qué entendía el pobre usuario, por ejemplo el responsable de préstamos de la sucursal de Antequera, cuando llamaba, medio llorando, para decirte que no podía abrir un préstamo hipotecario porque la máquina no le dejaba... y recibía semejante *erudita* contestación.

Esta escasez de buenos profesionales trajo dos consecuencias, buenas o malas, según se mire: **Se activó el mercado** para estos profesionales (para nosotros, vaya), que comenzaron a cambiarse de empresa, explotando su conocimiento tan exclusivo y bien valorado, y por consiguiente **se comenzaron a mover los sueldos**, no sólo en las empresas que fichaban nuevos empleados, sino también en las que pretendían conservar los suyos... o sea, en todas. Y este movimiento salarial fue bueno para nosotros, los informáticos del momento. Ganábamos bastante dinero. Trabajábamos mucho, eso es cierto también, pero el sueldo era muy bueno y nosotros éramos buenos, importantes, imprescindibles...

... *Y nos endiosamos.*

Sí, lo reconozco. Lo sé de buena tinta porque *yo también fui un dios...* menor, quizá, pero dios, al cabo. Tan importantes nos sentimos, tan insustituibles, tan necesarios... que nos volvimos unos perfectos cretinos. Nos olvidamos de lo más importante (de *lo único importante*): que no somos ni más ni menos que un Departamento de Gasto, es decir, que el resto de la empresa nos ve como *Un Mal Necesario*.

Y, encima, ¡caro!

Nosotros, los informáticos, no generábamos un duro para el negocio: eran los sufridos comerciales, gestores y administrativos los que ganaban con sus ventas y su trabajo día a día el dinero suficiente como para pagar no sólo sus propios sueldos, sino los de los informáticos y los de todos los demás, así como los dineros que se reparten a los accionistas.

Sigo reconociéndolo: Nos encastillamos en nuestra *torre de marfil* y llegamos a pensar que **lo único importante** era que nuestros sistemas fueran como la seda (*desde nuestro punto de vista, eso sí*), que el tiempo de respuesta fuera bueno, que no fallaran las aplicaciones y, sobre todo, que *nos molestaran lo menos posible con peticiones, cambios, problemas y demás zarandajas*.

Las peticiones de cambio en alguna aplicación por parte del usuario, que hasta hacía poco habían sido atendidas con prontitud y dedicación, comenzaron a ser sistemáticamente distraídas, retenidas, paradas, postergadas, cortocircuitadas, ignoradas... Las excusas eran... bueno, eran para estar allí y oírlas. Un ejemplo cualquiera:

“*Mira: para hacer esta modificación que pides, debo convertir un índice Physical-Child-First en un Physical-Child-Last, con lo que debo modificar cuarenta y cinco transacciones y treinta y siete programas batch, hacer un REORG y parar la Base de Datos una semana y media, y además las transacciones de consulta por LTERM tendrían un tiempo de respuesta mucho peor que ahora porque se produciría un encolamiento estocástico masivo y afectaría al rendimiento ciclotímico del sistema transversal esporádico del VSAM, y bla, bla, bla...*”.

...Y el pobre usuario, compungido y desarbolado, te pedía perdón por habersele ocurrido semejante proposición deshonestas, te invitaba a un café con churros como desagravio... y se volvía a sus Cuarteles de Invierno completamente frustrado, porque en realidad él o ella seguía pensando que *poder sacar los movimientos de las cuentas ordenados por fecha de valor en vez de por fecha de operación* no debería ser tan complicado...

Las aplicaciones se complicaban más y más... costaban más y más... y cada vez estábamos más lejos de los usuarios. Los árboles no nos dejaban ver el bosque. *Y al final lo pagamos*. Pero ésa es otra historia y será contada en otro momento.

Yo creo que de esta época (principios y mediados de los ochenta) fue cuando empezó a aparecer ese *sano odio* que la mayoría de usuarios de las empresas, y no sólo las grandes, tenían y siguen teniendo a los informáticos. Claro que a lo mejor es completamente natural el sentimiento de unos y de otros, porque es cierto que, hoy por hoy, casi ninguna empresa sería viable sin su informática, pero *más cierto aún es que, sin su empresa, casi ningún informático sería viable*.

Yo, por mi parte, confieso que siempre intenté dar en la medida de lo posible el mejor servicio a mis atribulados usuarios, ponerme en su lugar y mantener las aplicaciones lo más adecuadas y fáciles de utilizar posible, eso que ahora, años más tarde, se llama

*usabilidad*. Pero también he de entonar el *mea culpa*, pues soy culpable de una buena parte de los pecados descritos.

Muchos años después, he de reconocer que una parte de los de la profesión sufrimos (en *pasado*) y aún sufrimos (en *presente*) esta enfermedad. Si mis pobres palabras sirven para que algunos de vosotros, queridos lectores informáticos, reflexionéis un poco sobre esta circunstancia, habrán cumplido su misión. **Y os pido humildemente perdón, si en algo os he ofendido...**

Volvamos ya a las Bases de Datos, que esto va de Base de Datos, no de *golpes de pecho*...

Ya a principio de los setenta había algunos adelantados que pensaban que **esta generación de Bases de Datos podría abocarnos, con el tiempo, a un callejón sin salida.**

Era necesario un **método de diseño más sencillo**, pues cada vez más y más aplicaciones se escribirían en todo el mundo y no era conveniente que todas y cada una de ellas necesitaran unos técnicos especializados y muy expertos, cada uno de ellos en un producto diferente e incompatible con el resto.

Era necesario **solventar la falta de compatibilidad entre aplicaciones** mediante una interfaz común.

Era necesario encontrar alguna cosa, algún sistema para facilitar **la elección del mejor camino para recuperar la información pedida** sin necesidad de tener que conocer de antemano y programar cuidadosamente el mejor camino para hacerlo, es decir, poder realizar la navegación de forma automática.

Era necesario que **un simple cambio en la definición de la Base de Datos no obligara a modificar todos los programas** que acceden a ella.

Era necesario, por fin, **depender cada vez menos de nosotros, los informáticos.**

En una palabra, era necesario **volver a acercar la tecnología al negocio**, porque en los últimos tiempos ambos se estaban separando cada vez más, y la pinta era que podrían llegar a perderse de vista...

Para evitarlo en lo posible hubo que recurrir a un drástico cambio en la tecnología, que dio origen al advenimiento de las Bases de Datos Relacionales, que son las que ostentan el cuasi-monopolio en la actualidad.

## 11 - La llegada de las Bases de Datos Relacionales

Tras el repaso con cierta profundidad de cómo fue el comienzo de las Bases de Datos jerárquicas o en red y ver cómo estaba a mediados de los ochenta el panorama de la tecnología de almacenamiento y recuperación de la información, veremos ahora cómo fue la aparición, a mediados de esa misma década, de nuestras amigas las Bases de Datos Relacionales, sin las que hoy en día no podríamos entender la informática.

Insisto una vez más: no intento realizar una crónica oficial de nada, sino contar mis recuerdos y reflexiones de aquella época, la segunda mitad de los ochenta, en que también viví muy de cerca el lanzamiento inicial, los primeros balbuceos y la final adopción, no sin reticencias, de estas primeras versiones *relacionales* de las bases de datos.

Efectivamente, a mediados de los ochenta existían múltiples bases de datos, jerárquicas o en red, tanto de fabricantes como de compañías de software independiente, todas ellas mayormente incompatibles entre sí... Pero **pronto iban a comenzar a cambiar las cosas**.

Pero... Remontémonos en el tiempo...

A principios de los setenta, un ingeniero de IBM en San José, California, el británico **Edgar. F. Codd**, estaba convencido de que las Bases de Datos no tenían mucho futuro en la forma que estaban planteadas, sobre todo desde el punto de vista del diseño. Diseñar correctamente una Base de Datos Jerárquica o en Red no era nada sencillo (doy fe) y programar correctamente los accesos, menos aún (doy fe, también). Él abogaba por un método de diseño mucho más natural, más sencillo y que viniera determinado exclusivamente por los datos de la Aplicación.

Si os acordáis del capítulo sobre la programación estructurada, tanto el francés Warnier como, sobre todo, el inglés Jackson estaban pensando casi lo mismo: que para diseñar programas la única fuente fiable de información eran **los datos** sobre los que se aplicaba dicho programa.

Así que, tan pronto como en 1970, Codd publicó un *paper* de título “*A Relational Model of Data for Large Shared Data Banks*”, en el que expresó sus teorías. Fijaos: “Data Bank”. Por entonces, y durante varios años, el término que se usaba preferentemente era “Banco de Datos”. El caso es que Ted Codd no tuvo mucho éxito: IBM, su patrono, no puso ningún interés en meterse en más berenjenales.

Sinceramente, a mí no me extraña nada, puesto que, por un lado, estaba el reciente lanzamiento de IMS, en 1968, base de datos jerárquica en la que IBM había hecho un gran esfuerzo inversor y que pretendía, lógicamente, amortizar con creces; y, por otro, con la tecnología de la época era imposible imaginarse siquiera la implementación de algo como un Gestor Relacional de Bases de Datos.

Pero... **¿En qué consistían las ideas de Codd?**

En primer lugar, como ya he comentado, el diseño debe ser realizado de manera natural, utilizando la más sencilla posible de las representaciones de los datos, es decir, **los ficheros planos**, los secuenciales de toda la vida.

Nada de complicadas estructuras reticulares ni jerárquicas: basta con la Relación (la

lista) de todos los datos necesarios para la Aplicación: un compendio de todos los campos a gestionar en una única Lista de Datos.

De aquí viene el propio nombre de “*Relacional*”, puesto que se basa en Relaciones o listas de datos. En la ilustración, un mini-modelo relacional.



Además, la recuperación (toda la gestión, en realidad) de la información debería atenerse a las **normas formales del Álgebra y del Cálculo**. Entonces Codd describe el **Álgebra Relacional** aplicable a sus relaciones, álgebra que en realidad existía ya pero que no había sido muy explorada hasta entonces, que permite realizar operaciones con Relaciones, con Listas. Sus operaciones básicas son:

**Selección**, que permite obtener un cierto número de filas (*tuplas* en jerga relacional) de la Relación original, es decir, un subconjunto de las filas originales.

**Proyección**, que permite extraer un determinado número de datos -columnas- de una determinada Relación.

**Producto Cartesiano**, que permite obtener todas las combinaciones resultantes de la mezcla de dos Relaciones con columnas diferentes (pero con clave idéntica).

**Unión**, que permite obtener una única relación que contenga todas las filas de dos o más relaciones con las mismas columnas.

**Diferencia**, que permite obtener las filas de una relación que no tienen correspondencia en otra relación, es decir, las que están en una, pero no en la otra. Y finalmente:

**Renombrado**, añadida posteriormente por cuestiones formales, que permite cambiar el nombre a una columna de una relación.

Si os acordáis de la Lógica de Conjuntos, quizá echaréis en falta la **intersección**, pero formalmente no es necesaria, pues es el resultado de aplicar dos diferencias consecutivas: si los conjuntos a tratar fueran A y B, la intersección de A y B se calcula como  $(A - (A - B))$ .

En fin, el Álgebra Relacional queda definida como un subconjunto de la Lógica de Primer Orden. Y el Teorema de Codd establece la correspondencia entre el Álgebra Relacional y el Cálculo Relacional.

La consecuencia es que, estando como está toda la Teoría Relacional gobernada por el Álgebra Relacional (o el Cálculo, que en realidad es lo mismo) y, en definitiva, con la Lógica clásica y, por tanto, con el bien conocido **Cálculo de Predicados**, *es posible establecer procedimientos puramente matemáticos para obtener el resultado de una petición a partir de la o las Relaciones Originales y la enumeración de los resultados deseados*.

En una palabra: sería factible **construir un programa que**, dados los resultados deseados (o sea, las especificaciones funcionales), y la estructura de la Base de Datos, calcule **matemáticamente el mejor camino para poder obtener la información...** eliminando con ello, de golpe, uno de los puntos más espinosos de la programación de las Bases de Datos tradicionales: la necesidad de que un programador avezado escribiera el código exacto para optimizar los accesos. Estaríamos así cerca de conseguir por fin el

programador automático.

Se trata, en efecto, de esa procelosa e insondable pieza de código que es conocida como **Optimizador**. Pero no nos adelantemos...

Porque, incluso en 1970, era evidente que almacenar *toda la información* de una gran Aplicación en un solo fichero físico era entonces, como lo sigue siendo ahora, una auténtica locura. La gigantesca redundancia de información que se produciría haría inviable cualquier posible proceso. Así que Codd empezó a describir Reglas para Normalizar Bases de Datos Relacionales. Ya en 1971 había formalizado la primera, y tan pronto como en 1973 estaban ya formalizadas las **Tres Primeras Formas Normales**, que es hasta donde todo el mundo llega normalmente en el proceso de diseño, o donde todo el mundo se queda, según lo veáis.

En 1974, entre Codd y Raymond Boyce definieron la así denominada *Forma Normal de Boyce-Codd* y posteriormente se definieron la *Cuarta*, la *Quinta* e incluso la *Sexta Formas Normales*, esta última tan tarde como en 2002.

A fuer de ser sincero, no me preguntéis para qué sirven estas últimas, ni si tienen mucha aplicación real: en mi aburrida vida de diseñador de aplicaciones comerciales jamás me he visto en la necesidad de llegar más allá de la famosa Tercera Forma Normal...

Aunque IBM, la *Madrastra* de los *Enanitos* de principios de los años 70, no tuvo el menor interés en abrir otro melón, pues bastantes tenía ya abiertos por entonces, el amigo Codd era un tipo *machacón*: con el tiempo consiguió que IBM pusiera fondos para arrancar su primer proyecto relacional, el tatarabuelo de todos ellos: **System/R**.

Sin embargo, por algún motivo ignoto (quizá porque resultaba demasiado “*matemático*” para las entendederas de los programadores de la época, yo el primero, porque por tema de patentes, desde luego, no era), en lugar de utilizar el Lenguaje **ALPHA** que Codd había diseñado, inventaron otro diferente, de nombre **SEQUEL** que, años después y debido a que el nombre **SEQUEL** estaba registrado ya, cambió su nombre a **SQL**... un nombre que quizá os suene de algo.

System/R terminó con cierto éxito, no porque fuera muy eficiente, que no lo era ni de lejos, sino porque demostró que *era factible* construir un Gestor de Bases de Datos Relacionales con un buen rendimiento en entornos transaccionales. Y muchas de las decisiones de diseño que se tomaron entonces han sido adoptadas por todas las Bases de Datos Relacionales.

Por ejemplo, almacenar la propia definición de las Bases de Datos (Tablas, Índices, etc., en definitiva, los metadatos o datos sobre los datos) en unas tablas más, unas tablas mondas y lirondas, el famoso **Catálogo**, y acceder a ellas como si fueran unos datos cualesquiera, está ya en el diseño de System/R y todos sus seguidores siguieron la misma solución, que será muy bonita desde el punto de vista formal, pero es una solución espantosa desde el punto de vista del rendimiento; en IMS, por ejemplo, las definiciones de las Bases de Datos, índices, accesos, etc., se compilan (en realidad, *se ensamblan*), generando un código compacto y muy eficiente que es el que es usado por el Gestor.

Y otra decisión que tuvieron que tomar los diseñadores de System/R fue cómo realizar la comunicación del resultado de una petición de información a la Base de Datos y los programas que usan esta información. El motivo es que la propia lógica relacional es *per se* una *Lógica de Conjuntos* y, por tanto, cada query puede devolver cero filas, o una... **o muchas**, incluso todas las filas de una tabla o conjunto de tablas. Y, claro, todos los ordenadores siguen siendo de momento unas humildes y escalares máquinas *von Neumann* que tienen la costumbre de tratar secuencialmente los datos y no todos a la vez, de golpe.



En una palabra, había que hacer convivir el hecho de que las queries relacionales son del tipo **Set-At-A-Time**, mientras que las máquinas eran entonces, y por el momento siguen siendo, del tipo **Record-At-A-Time**... Y se inventaron la técnica de los **Cursores**, con todas sus conocidas sentencias asociadas (DECLARE CURSOR, FETCH, OPEN, CLOSE...), que permiten tratar un conjunto de filas o de datos como si se tratara de un fichero secuencial mono y lirondo. También los Cursores están presentes en prácticamente todas las Bases de Datos Relacionales de hoy en día.

System/R incluso llegó a instalarse en algún cliente, sobre 1977-78, pero tuvo escaso éxito, debido a su inexistente rendimiento en las máquinas de la época.

Pero el caso es que también había *vida relacional* fuera de los esfuerzos (aunque no demasiado esforzados todavía) de IBM.

En la Universidad de Berkeley, California, habían asumido los planteamientos de Codd y comenzaron un proyecto de investigación que dio origen, a principios de los ochenta, a la Base de Datos **Ingres**. Fue un proyecto bajo licencia “BSD” y, por tanto, por una pequeña cantidad se podía adquirir el código fuente y reutilizarlo. Buena parte de las Bases de Datos de hoy en día deben mucho a aquel Ingres primigenio: Sybase, Microsoft SQL Server y NonStop SQL, entre otras.

Mientras tanto, IBM seguía trabajando en las secuelas de System/R. En 1982 sacó al mercado la Base de Datos **SQL/DS**, exclusivamente para DOS/VSE. Este VSE es un Sistema Operativo de IBM “competidor” de MVS y heredero de DOS, el primer sistema operativo basado en disco de IBM. Todavía hoy en día siguen existiendo por ahí instalaciones funcionando en VSE, y tan ricamente...

A partir de las especificaciones públicas de SQL/DS, Larry Ellison, fundador de Relational Software, y un equipo de ingenieros, entre los que había alguno que participó en el proyecto System/R, diseñaron una Base de Datos llamada **Oracle**, que comenzó a ser vendida alrededor de 1981 u 82, en principio para Digital/VAX, aunque la primera versión “utilizable” se vendió a partir de 1983-84, los mismos años en los que IBM lanzó **DB2**, el equivalente de SQL/DS, pero para MVS, sólo en entorno mainframe, desde luego.

El nombre “**DB2**” se eligió para enfatizar la idea de que IMS era *la primera* Base de Datos (DL/1 era el lenguaje de tratamiento de la información), y DB2 era *la segunda* (y mejor) de ellas...

Igual os sorprenderá cómo era posible que otros fabricantes usaran de forma abierta las especificaciones de Codd, ingeniero de IBM, quien lógicamente patentaría cualquier descubrimiento, estructura o método descubierto por sus empleados... Pues en este caso, independientemente de que IBM patentara todo lo que pudo, resulta que Codd participó en un cierto Congreso en el que proporcionó a los asistentes unos *papers* con su intervención, en los que resumía todos sus hallazgos en teoría relacional. Y esos *papers*, al haber sido divulgados públicamente, no pudieron ser objeto de patente... eso, al menos, me contaron en IBM hace muchos, pero muchos años; ignoro si ésa es la verdad, toda la verdad y nada más que la verdad.

La versión DB2 v1.0 teóricamente funcionaba... siempre que las tablas tuvieran algunos cientos de filas como máximo, no se hicieran muchos joins o, mejor, ninguno, y no hubiera que recuperar nada ante fallos. En una palabra, era completamente inservible para Aplicaciones en Producción. Larry Ellison fue más listo: la primera versión de Oracle lanzada al mercado era la 2.0... aunque le pasaban igualmente cosas muy parecidas.

A la versión 1.0 de DB2 siguió un año después la v1.1. Ésta fue la primera versión

que probamos en el banco en que trabajaba por entonces. La presión comercial de IBM y las facilidades de apoyo que ofrecía en la instalación y prueba hicieron que estas primeras versiones se instalaran en muchísimas de las instalaciones españolas de MVS en aquellos años tempranos (86-87-88). La conclusión general es que el nuevo DB2 era un producto que efectivamente *prometía*, pero que no se podía usar aún para Producción real.

Un ejemplo que yo viví: una de las pruebas que hicimos fue pasar un proceso de actualización a una tabla de algunos miles de filas, no más, que actualizaba todas las filas de la tabla poniendo un cierto valor a una o varias columnas, y entonces, a mitad del proceso, cascar deliberadamente el programa: le hacíamos dividir un número por cero, cosa que a casi ningún ordenador le gusta demasiado: en MVS obtienes un hermoso *abend SOCB*.

La Base de Datos debía, entonces, acudir al log y recuperar todos los cambios para dejar la tabla como al principio del proceso. Y eso fue lo que *comenzó* a hacer... Un par de horas más tarde, aburridos, cancelamos también el recovery... y entonces *comenzó a recuperar la propia recuperación*... hasta que otras tres horas después, paramos todo el DB2, que de todos modos se había quedado poco menos que frito y lo mandamos todo a hacer gárgaras. Desde luego, muy estable no era aquello.

Como todas estas cosas se supieron más temprano que tarde, la consecuencia es que nadie instaló tampoco la versión 1.1 de DB2. Pero había, eso sí, gran ruido e interés en esta nueva tecnología.

Además de los esfuerzos de los fabricantes por convencernos de las supuestas bondades de la *cosa relacional*, hay que resaltar aquí la enorme labor evangelizadora realizada por Chris Date, dando conferencias por todo el mundo (en España también estuvo), escribiendo libros, artículos... Una buena parte de su éxito inicial es debido a él.

Tanto interés había entonces que el Grupo de DB2 de GUIDE se convirtió en el más numeroso con diferencia de todos los grupos de GUIDE, en España, al menos. Por si no sabéis qué es GUIDE, se trata de una Asociación de Usuarios auspiciada por IBM en el que los usuarios participan, junto con técnicos de la propia IBM, para poner en común las experiencias que cada cuál tiene con los productos de esa marca, de hardware y de software: MVS, CICS, IMS, etc., así como hacer peticiones sobre funcionalidades de las nuevas versiones... y charlar de sus cosas. Cuando se constituyó el Grupo de DB2 casi todo el mundo se apuntó, pues, en la práctica, todas las grandes instalaciones sabíamos que, tarde o temprano, *acabaríamos con uno o varios DB2's en nuestra instalación*.

El motivo de tanta expectación era, sobre todo, la esperanza de que por fin íbamos a poder disfrutar de una herramienta con una **interfaz común**, el SQL, que permitiría migrar aplicaciones de entorno con facilidad y sencillez (aunque no se hiciera casi nunca) y, sobre todo, que **permitiría reaprovechar** (*¡por fin!*) **la formación de los técnicos**: una vez sabes SQL de una determinada Base de Datos, puedes manejarte con cualquier otra Base de Datos Relacional... más o menos.

Porque nunca nos creímos del todo eso de que el Optimizador sería cada vez más listo y sería capaz siempre, pero siempre, de encontrar el camino más eficaz para resolver cada query sin necesidad de *ayudarle*. Ya sabéis que, al menos hasta ahora, casi treinta años más tarde, teníamos razón: los Optimizadores son cada vez más listos... y siguen metiendo la pata con demasiada frecuencia como para poder dejarlos siempre solos; de ahí que en ocasiones haya que darles “hints” para orientarlos.

IBM comenzó también a dar formación para los incipientes “probadores”, y así hasta que a principios de 1987 IBM sacó al mercado la versión v1.2. Esta versión ya

funcionaba razonablemente bien, pero aún tenía un problema: ¡no hablaba con IMS/DC, sólo con CICS! Y esto era un serio inconveniente para aquellas instalaciones, las más grandes, que teníamos IMS no sólo como Base de Datos, sino como Gestor Transaccional... porque nadie estaba dispuesto a cambiar el robustísimo IMS/DC por un CICS que, en aquellas épocas, era mucho menos potente, más inestable y tenía un rendimiento muy inferior.

Por fin, a fines de 1987 o principios de 1988, IBM publicó la versión v1.3, que, además de seguir mejorando su funcionalidad y su rendimiento, era compatible con el IMS. Seguía teniendo importantes lagunas. Por ejemplo, la compilación de las queries tardaba muchísimo y además producía espeluznantes encolamientos en el acceso al Catálogo; el rendimiento de los joins seguía siendo malo (como ahora, vaya); un recovery de un proceso batch en base al log seguía tardando mucho, aunque al menos, a partir de entonces, siempre acababa bien... pero obviando estos problemas conocidos, que podían solucionarse a golpe de normativa prohibiendo *por Decreto* hacer lo que se sabía que no iba a ir bien, daba un rendimiento suficientemente bueno como para poder confiar en ella.

Ése fue, por fin, el momento en que, en el plazo de unos pocos meses, muchas empresas, la mía entre ellas, tomaron la decisión de **comenzar a usar DB2 “en serio”**, es decir, para las nuevas aplicaciones que se comenzaban a escribir... y para ello, hubo que preparar extensísimos Planes de Formación para todo tipo de profesionales de la informática: Analistas Funcionales u Orgánicos, Programadores, Técnicos de Sistemas, Operadores...

El Departamento de Formación de IBM quedó desbordado, pues no había tantos buenos formadores en un producto tan nuevo como lo era DB2 y los clientes tuvimos literalmente que hacer cola, porque no había otra alternativa: ninguna empresa de formación informática tenía entonces conocimientos de DB2 (ni de ninguna otra Base de Datos Relacional, diría yo) como para poder formar a nadie. Y claro, cuando una gran empresa decidía entrar de lleno en una tecnología básica como lo era ésta lo mismo debía dar formación a cuatrocientas o quinientas personas...

Sobre 1988 comenzamos por fin a escribir las primeras aplicaciones usando DB2. Hubo que reescribir buena parte de las normas de nomenclatura, diseño de programas, etc. Y hubo que asegurarse de que todas las incipientes aplicaciones que se escribían para DB2 fueran de una mínima calidad y que no hacían las cosas que se sabía que no iban bien... Por cierto, muchas de aquellas prohibiciones de fines de los ochenta siguen vigentes hoy en día en muchas instalaciones, a pesar de que lo que entonces iba mal ahora ya va bien... y es que *¡mira que nos cuesta trabajo cambiar a los informáticos!*

Así que éste fue el terreno abonado para la tercera gran explosión tecnológica de la época: la adopción de las Arquitecturas, las Metodologías de Desarrollo y las Herramientas CASE, cosas muy interesantes de las que hablaremos en los próximos capítulos.

Ahora, permitidme solamente unos breves apuntes sobre cómo es la compilación de programas que accedan a DB2. Es posible, desde luego, lanzar una query desde un programa Cobol, o del lenguaje que sea, y que el Optimizador la reciba, la analice, la compile hallando el mejor camino posible para su resolución y, por fin, que la ejecute y envíe el resultado al programa llamador.

Pero es un proceso costoso, debido, entre otras cosas, a que toda la información acerca de las propias Bases de Datos está contenida en otras Bases de Datos, lo que provoca una fuerte concurrencia sobre el Catálogo, y por tanto no es nada aconsejable hacer ese

“**SQL Dinámico**”, ni en batch ni online.

Así que IBM inventó para DB2 el “SQL Estático”, es decir, **las queries se compilan y optimizan en batch**, mediante un proceso denominado “**BIND**”, que compila cada query, calcula el mejor camino a usar en función de la propia query, del número de filas de cada tabla, la distribución de las claves, los índices existentes y lo desorganizados que estén... y esa información del “mejor camino”, denominada **Plan**, se almacena en el propio catálogo y se usa siempre... para bien o para mal. Técnica ésta, la del SQL Precompilado, que, a pesar de que según la teoría no es recomendable en absoluto, casi todas las Bases de Datos actuales utilizan... por algo será.

El proceso de “BIND”, pues, se convierte en una parte más de la compilación de cualquier programa que acceda a DB2, que queda, por tanto, con cuatro pasos, por lo menos:

**Precompilación DB2** (este paso convierte las llamadas a DB2 codificadas con “EXEC SQL” en llamadas a módulos internos de DB2 que resuelven cada petición particular);

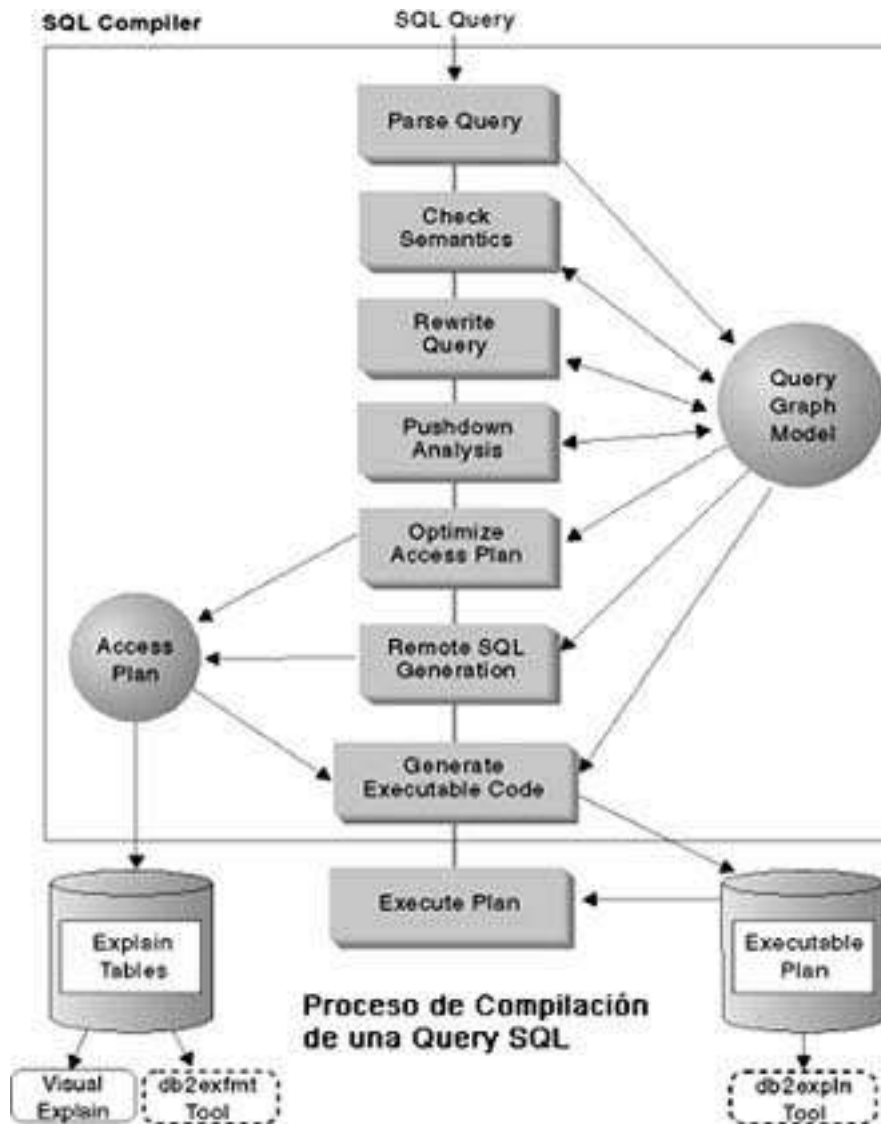
**Compilación** (por ejemplo, en Cobol, lo que genera el programa objeto);

**Linkedit** (genera el programa ejecutable, uniendo todos los módulos llamados por el principal, incluidos los de DB2), y

**BIND** (que obtiene el mejor camino para cada query).

Si además el programa accede a CICS, tendrá también su propia Precompilación, que en caso de IMS no es necesaria. En una palabra, la compilación comienza a ser un proceso pesado y muy consumidor de recursos...

A continuación tenemos un diagrama del proceso de compilación de una query en DB2 o, con mínimas diferencias, en todas las bases de datos relacionales.



Este sistema soluciona el problema, siempre que las condiciones de las tablas no varíen, al menos en exceso, es decir, que se creen nuevos índices o se borre alguno existente, que la tabla se desorganice debido a las inserciones y borrados reiterados, o simplemente que, debido al devenir normal de los acontecimientos, crezca o decrezca apreciablemente el tamaño de las tablas. Cuando alguna de estas cosas ocurre, es necesario hacer “REBIND” de todos los Planes que afecten a la tabla, para permitir a DB2 calcular de nuevo el mejor camino... o no, pero bueno.

Además, también era posible acceder a las tablas DB2 directamente desde TSO, sin necesidad de programar, mediante dos productos específicos: **SPUFI** y **QMF**.

Estos productos permiten a los usuarios finales lanzar sus propias queries contra las Bases de Datos y obtener los resultados; ambos productos, sobre todo QMF, fueron pilares básicos para la introducción, a fines de los ochenta, del concepto de **Centro de Información**, del que algo comentaremos cuando más adelante le llegue su turno al Data Warehouse.

IBM concentró sus esfuerzos *relacionales* de la década de los ochenta en el mundo mainframe... donde arrasó (y también en el mundo AS/400, para el que sacó al mercado

DB2/400, base de datos que comparte con DB2 el nombre, casi toda la interfaz SQL... y poco más). No creo que hoy en día subsistan muchas Bases de Datos Relacionales en entorno mainframe que no sean DB2...

Pero en los años ochenta IBM dejó de lado el *resto de los mundos*. De hecho, recordad que hasta 1990 no anunció IBM su entrada en el mundo UNIX, de la mano de la gama RS/6000. Y esos “otros mundos” fueron aprovechados por pequeñas empresas independientes que lanzaron rápidamente productos que funcionaban en VMS (el Sistema de los VAX de Digital), en UNIX, en OS/2, e incluso en MS/DOS. Yo tuve la oportunidad de probar la versión de Oracle para MS/DOS allá por 1990 o así: os podéis imaginar que no servía para nada, aunque las queries las hacía, y realmente las hacía bien... siempre y cuando las tablas no tuvieran más allá de quince o veinte filas.

Estas compañías fueron, sobre todo, Relational Software con su producto **Oracle**, aunque pronto cambió su nombre a *Oracle Corp.*, **Informix** (acrónimo de Information on Unix), y **Sybase**, fundamentalmente, aparte de la propia Ingres, que sacó al mercado su versión UNIX a mediados de la década.

El gran avance de todas estas Bases de Datos lo constituyó el hecho de que todas ellas tuvieran la misma o casi la misma interfaz, **SQL**, que copiaron de la que IBM inventó pero *no patentó*, o quizá *no pudo* patentar: de ahí su gran éxito, como ocurrió con los PC's. Por una parte les ahorró costes de diseño y desarrollo; por otra parte, les ayudó comercialmente a crear “masa crítica”. Todas ellas eran compañías pequeñas, que un buen día podían quebrar... pero la inversión realizada por los clientes podría ser fácilmente transportable a otra de las que sobrevivieran; este hecho facilitó indudablemente la toma de decisiones a favor de esta tecnología relacional.

Mi opinión es que IBM se equivocó de estrategia al dejar durante unos años desatendido este mercado (bueno, no sólo yo, también IBM lo cree... *ahora*), mercado que creció mucho más rápido de lo que pudieron prever. Y ahí está ahora Oracle, por ejemplo, como una de las mayores compañías de tecnología de mundo, mayor que IBM, de hecho, que además, en abril de 2009, adquirió Sun Microsystems, entrando de lleno, por primera vez, en el negocio del hardware...

Este no fue, desde luego, el único error de estrategia que IBM cometió, pero ésta es otra historia y será contada en otro momento.

Fueron apareciendo, además, otros nichos de mercado que IBM no atendía: en el mundo de los sistemas tolerantes a fallos (ahora casi todos los sistemas importantes tienen una gran tolerancia a fallos, pero no entonces), Tandem se hizo con una buena parte del mercado de aquellos Sistemas Críticos que no podían dejar de funcionar por un simple fallo de un componente cualquiera, hardware o software.

Por cierto, IBM tenía también, cómo no, oferta en este mercado: el IBM System/88, que en realidad eran ordenadores Stratus vendidos por IBM como OEM... aunque fue Tandem quien lideró el mercado durante esos años. Y Tandem puso en el mercado NonStop SQL, Base de Datos Relacional para sus sistemas tolerantes a fallos construida a partir de la base de Ingres.

Otro nicho lo ocupó otra pequeña compañía, Teradata, que comenzó a vender la primera **Máquina de Base de Datos**, es decir, un hardware diseñado específicamente para obtener el máximo rendimiento en accesos a los discos magnéticos y una Base de Datos relacional especializada que permitía usar eficientemente el hardware paralelizando accesos e incrementando en mucho su velocidad, sobre todo en lectura, y, de paso, permitiendo

incrementar el espacio en disco disponible en los mainframes, ya que Teradata tiene desde su creación una conectividad excelente con estos mainframes: se conecta vía canal, como un dispositivo externo más, y para MVS se trata de una cinta magnética... por lo que instalarlo no tiene apenas trabajo que hacer en el sistema principal.

Teradata, de todos modos, encontró su nicho de mercado (siendo precisos, prácticamente *lo creó*) cuando a mediados de los noventa comenzó a ponerse de moda, o mejor, *a necesitarse* por todas las compañías, el concepto de Data Warehouse. Pero ésa es otra historia y será contada en otro momento, en concreto, en los capítulos dedicados al Data Warehouse, el Business Intelligence, etc.

Por fin, los fabricantes de otras bases de datos que no eran ni mucho menos relacionales comenzaron a añadirles una interfaz relacional para adaptarse a los nuevos tiempos. Datacom y Adabas fueron dos de ellas. El rendimiento era peor, sin duda, que utilizar su interfaz tipo CALL de toda la vida... pero así tenían más *appeal* para su venta basándose en la portabilidad de aplicaciones, etc. No les fue mal del todo: ambas siguen existiendo en nuestros relacionales días, lo que es mucho decir.

En una palabra, aparecieron multitud de Bases de Datos Relacionales, pseudo-Relacionales o no-Relacionales-con-interfaz-SQL, con lo que la confusión del mercado creció y creció...

Así que nuestro buen amigo Codd, siempre velando por la *pureza relacional*, se aprestó a sacarnos de dudas: publicó sus famosas **12 Reglas** para poder discernir las Bases de Datos Relacionales *de verdad*, de las que sólo *decían que eran* relacionales, pero no lo eran. Para los que tengáis curiosidad, *ninguna* Base de Datos de principios de los noventa cumplía con las 12 reglas, ni tan siquiera DB2.

Ignoro si el cumplimiento o no de las reglas famosas se tuvo en cuenta a la hora de hacer la selección de una u otra Base de Datos para un proyecto concreto por una empresa concreta. Pero sí que se hicieron estudios sobre cuál era la que más o la que menos cumplía con las dichas reglas.

Recuerdo que, a principios de los noventa, cierta Universidad madrileña anunció, en el entorno del **SIMO**, la feria sobre informática más importante de España durante mucho tiempo, una charla en la que nos explicaría un estudio completísimo que habían realizado sobre cuál era la mejor Base de Datos Relacional. Aprovechando que por aquella época aún iba yo al SIMO todos los años (hace ya muchos que no, y por fin desde el año 2008 no ha ido al SIMO *ni siquiera el propio SIMO*... porque fue cancelado), fui a la charla de conclusiones. Acudimos muchos profesionales, pensando... “Vaya, por fin un estudio *serio, independiente y español* sobre Bases de Datos, ¡pardiez, qué interesante!”.

Pues no.

Resulta que habían realizado *una encuesta* a los doce o catorce vendedores de RDBMSs que entonces había en España, mediante el envío de un cuestionario en el que estos contestaban amablemente y sin casi mentir cuál era el grado de cumplimiento de su Gestor sobre las doce reglas de Codd.

Por ejemplo: “¿Su Base de Datos mantiene un Catálogo Dinámico Online basado en el modelo relacional? Conteste Sí, No, o *Según el día*”, y así con todo. Y ya.

...Y en base a las respuestas, con un sencillo proceso de tabulación (porcentajes de cumplimiento y poco más, nada de estadísticos complejos, ni siquiera una triste desviación estándar), llegaron a la conclusión de que la mejor de todas, la *chachi piruli*, era CA/Universe, de Computer Associates, que por entonces estaba empeñado en fusionarla con la otra Base de Datos de su propiedad: Datacom, aunque ignoro si llegaron a terminar

la fusión alguna vez...

Este viejo informático, aunque entonces no lo fuera tanto (tan *viejo*, quiero decir, que informático sí que lo era...), pero que siempre ha sido un poco *preguntón*, no pudo resistirse, e hizo dos preguntas en el turno de Ruegos y Preguntas. Dos y sólo dos:

*Una:* «¿Han hecho algún tipo de prueba de funcionamiento en la práctica?»

*Respuesta:* «**No, ninguna.** La pobrecita Universidad no tiene fondos suficientes, *tralarí, tralará...*».

*Dos:* «¿Han utilizado, al menos, los manuales *técnicos* de las Bases de Datos para hacer la evaluación?» *Respuesta:* «Uy, no, menudo trabajo, pues vaya, todo en inglés...

**Nos hemos basado exclusivamente en las respuestas que los fabricantes han dado a las doce preguntitas del cuestionario.** Es que la Universidad tiene pocos recursos, ya sabe, *tralará, tralarí...*».

Y no hubo una pregunta *tres*, como es obvio: me levanté y me fui; a ver qué iba a hacer yo allí perdiendo el tiempo... Sinceramente, esto me dio mucha pena. Cuando lo comparo con la Universidad de Berkeley, arrancando un proyecto de investigación y creando de la nada Ingres... pues eso, que me da mucha pena. Y estoy yo bastante seguro de que las cosas no han mejorado mucho desde aquellos años hasta ahora, desgraciadamente.

Bueno, para acabar esta historia, IBM entró por fin en el mercado de Bases de Datos Relacionales para UNIX... perdón, para AIX, que era el UNIX que IBM se inventó para su gama RS/6000 y sacaron al mercado, sería sobre 1991 o 1992, **DB2/6000**. Que no tenía nada que ver con DB2, el del mainframe, ni con DB2/400, menos el nombre y la mayor parte de la interfaz.

La versión mainframe fue durante mucho tiempo responsabilidad del Laboratorio de Santa Teresa (California), mientras que DB2/6000 lo era del Laboratorio de Toronto (Canadá)... así que, para no perder la costumbre, los laboratorios parecían peleados y los planes de desarrollo de una y otra Base de Datos eran no sólo distintos sino, en ocasiones, contradictorios.

Este DB2/6000 con el tiempo se convirtió en lo que es hoy: **DB2 Universal Database**, y ya funciona en todos los sabores, olores y colores de UNIX, OS/2, Linux, supongo que Windows, PDAs, etc... menos en mainframe, donde sigue funcionando una cosa llamada *DB2*, sí, pero que no tiene mucho que ver con su *Universal* primo.

Y colorín, colorado, las Bases de Datos Relacionales se hicieron con el mercado. Y esta relacional historia, por cierto, se ha acabado.

Complementando los capítulos dedicados a las Bases de Datos de todo tipo, hablaré a continuación de eso tan bonito y lujoso llamado “**Ventana Batch**”, tan desconocido para muchos y que tantos dolores de cabeza nos ha dado a otros...



## 12 - La Ventana Batch

Tras repasar cómo comenzaron a ser lo que hoy en día son las Bases de Datos Relacionales, que sustituyeron a su vez a las Bases de Datos Jerárquicas o en Red, volveremos ahora nuestra vista, más concretamente, al Software, a las Aplicaciones que en ellas se basan. Vamos a ahondar en un tema común a todas ellas, pero que, por algún ignoto motivo, pocas veces se tiene en cuenta en la formación de las nuevas generaciones de informáticos: la **Ventana Batch**. Bueno, no me limitaré a hablar exclusivamente de lo que es la conspicua Ventana, sino también del propio diseño de los procesos batch.

Es exactamente lo mismo si hablamos de Bases de Datos Relacionales que de las No-Relacionales, con todas ellas habrá que tener en cuenta qué hacer cuando existen procesos batch que ejecutar contra esas Bases de Datos. Creo importante hablar de ella, y lo hago aquí porque estoy yo convencido de su importancia, siendo, como es, una gran desconocida.

Lo primero que hay que decir es que eso de la “Ventana Batch” comenzó a ser necesaria cuando se comenzaron a poner en marcha las primeras aplicaciones online. Es lógico; cuando todas las aplicaciones eran sólo batch, la “Ventana Batch” duraba todo el día...

En toda instalación grande se distingue claramente entre el proceso online y el proceso batch. Esto, que es tan obvio para los viejos rockeros, *en absoluto lo es para las nuevas generaciones*.

Casi todas las aplicaciones modernas tienen un gran (*de tamaño*) e importante (*de... ejem, importancia*) componente online, en el que se captura toda o casi toda la información, se atienden las consultas, se actualizan datos, se toman decisiones... Pero existe también la necesidad de realizar ciertos procesos que afectan, en lectura solamente o también actualizando, a una buena parte o a toda la Base de Datos.

Ejemplos hay en todas las áreas de negocio, pero usaré uno evidente: La **Liquidación de Cuentas**, en Banca, donde se calculan los intereses y comisiones de todas las cuentas del banco, o al menos las de cierto tipo, pero muchas, siempre muchas, lo que requiere tratar todos los movimientos del periodo a liquidar y generar los propios apuntes resultado de la liquidación, que se incorporan a la propia cuenta y a la contabilidad, se actualiza el saldo, se marcan como liquidados los movimientos tratados...

En fin, **se trata de un proceso muy pesado y complejo**, que lee y actualiza buena parte, o toda, la Base de Datos, que requiere tratar los movimientos ordenados por fecha de valor (pues de ninguna manera están así ordenados en la Base de Datos), y que suele durar horas. Esto no se puede hacer “online” (en realidad, no es que no se pueda, *es que es complicadísimo*), entre otras causas por:

**1 Los bloqueos.** Una Liquidación es el típico proceso que actualiza muchos (si no todos) los registros de una o varias Bases de Datos. Si simultáneamente existen transacciones online que actualizan esas mismas filas, se pueden producir infinidad de bloqueos, que, aunque se resuelvan todos correctamente, ralentizarían mucho a ambos

procesos: al batch y al online.

**2** Además, es posible que el proceso batch casque a mitad de su ejecución; entonces habría que recuperar los datos modificados por el proceso hasta el momento del error y dejarlos como antes de ejecutar nada, hasta revisar el motivo del fallo y solventarlo. Pero si simultáneamente se están ejecutando transacciones online... ¿qué saldo hay que dejar? Si una transacción ha utilizado unos dineros que han sido abonados por la Liquidación y ahora echamos ésta atrás... ¿qué hacemos con dicho saldo? Estos **problemas de integridad** son los más complicados de resolver.

**3** Luego, **el propio proceso de recuperación de los datos a partir del log es tremendamente pesado**: implica ir recorriendo dicho log hacia atrás, e ir reponiendo los datos previos a cada actualización... ¿toda una liquidación de cuentas? ¿De todas las cuentas? Mala idea...

**4** Pero es que, además, **la propia ejecución de procesos batch de cualquier tipo a la vez que el online genera problemas de integridad**. Por ejemplo, se puede hacer una transacción que implique actualizar dos registros diferentes de la Base de Datos, que están físicamente en lugares diferentes, digamos de dos oficinas distintas. Puede ocurrir que uno de estos registros haya sido ya modificado por el batch y el otro todavía no, porque aún no ha llegado hasta ese punto el programa, por lo que la decisión tomada por la transacción (verbigracia, permitir o no el pago) puede ser totalmente equivocada.

**5** Por fin, hay que tener en cuenta la **sencillez de los procesos**. En informática sencillez generalmente significa fiabilidad y rendimiento. Complicar innecesariamente los procesos no suele ser buena idea a largo plazo.

No sé si ha quedado claro, pero el punto más importante que el diseñador técnico tiene que tener en cuenta a la hora de diseñar un proceso batch no es *lo que tiene que hacer* dicho proceso, es decir, las especificaciones funcionales, sino **qué hacer si falla el proceso** (por un casque incontrolado, vaya, de esos que tontamente ocurren de vez en cuando). Así que la solución que todo el mundo toma siempre que puede para simplificar los sistemas, evitar interferencias y minimizar el impacto de un casque en Producción es **aislar los procesos batch del online**. Es decir, se mantienen las aplicaciones online funcionando durante la mayor parte del día, pero a cierta hora pactada con los usuarios, *se cierran*. Se cierran las aplicaciones, lo que significa que se paran las bases de datos de la aplicación y que no se pueden lanzar transacciones de esa aplicación, pero sí de otras: los gestores de teleproceso en sí nunca, o casi nunca, se paran.

Esas horitas en las que no hay online, distintas para cada aplicación, constituyen lo que se conoce como **Ventana Batch**. Ventana que, por cierto, cada vez es más pequeña, hoy es ya casi un ventanuco, *una claraboya*, pues debido a requerimientos de negocio cada vez deben estar más tiempo online las aplicaciones.

Lo primero que se hace tras parar el online, y por tanto parar las bases de datos, es **hacer copia de seguridad de dichas bases de datos**. Con esto quedan consolidados los cambios realizados por el online y se usa la copia como salvaguarda.

En el mundo mainframe de IBM, se suelen hacer "*Image Copies*", es decir, no se salva el contenido de la base de datos de forma lógica, registro a registro, sino que se hace

una copia física de los ficheros que forman la base de datos (datos, índices, etc), tal y como están: con sus huecos, sus registros borrados, sus desorganizaciones, etc. Pero es que así es mucho más rápido tanto el proceso de backup como el eventual de recovery. Para volver a dejar las Bases de Datos limpiatas, sin huecos, con los índices sin saltos, etc., se pasa periódicamente un proceso de “*REORG*”, que reorganiza la Base de Datos y la deja como los chorros del oro (básicamente lo que hace el REORG es descargar sus filas, luego ordenar los registros por clave y volverlas a cargar, esta vez convenientemente ordenadas)... hasta que arrancamos el online de nuevo y comienza de nuevo el espectáculo.

Bien, una vez obtenidas las copias de seguridad, se comienzan a lanzar los procesos batch en el orden que corresponda. Y una cosa que se deberá especificar es qué hacer si el programa (cada programa) falla. Pensaréis: *¡qué pesadito, con los casques...*! Los que estéis en esto, pensad si tengo o no razones para ser pesado.

En principio, ante un casque de un programa que actualiza una Base de Datos, hay dos alternativas: dejar que el Gestor de Base de Datos recupere el error de forma standard, usando el log de cambios, o recuperar de la última copia de seguridad, ignorando el log. Para esto último, por ejemplo, en IBM se define el fichero del log como DUMMY, es decir, que en realidad no existe, así que, al estar vacío, no se ha grabado realmente nada en él, ahorrando el tiempo de esa grabación y el Gestor de Base de Datos no tiene nada que recuperar. Entonces se vuelca (externamente, claro; suele ser el Planificador quien lo hace automáticamente) la última copia de seguridad sobre la Base de Datos, y listo. Tomar una opción u otra depende de muchas cosas: si es el primer proceso que actualiza o ya ha habido varios, en qué punto del proceso se hizo la última copia, cuántas filas actualiza de media el programa, etc.

**Los procesos batch deben pasarse en el orden lógico que han determinado los Analistas de la Aplicación**, que son los que saben bien cuáles son las relaciones lógicas entre procesos... o, al menos, se supone que les pagamos para que lo sepan, pero ésa es otra historia y será contada en otro momento.

Habrán procesos que podrán correr *en paralelo*; otros, no es posible, pues habrá **dependencias funcionales** entre ellos: por ejemplo, no se podrá pasar el proceso de Abono de Dividendo hasta que no se haya terminado el proceso de Consolidación de Carteras de Valores, que es el que determina finalmente cuántas acciones de qué compañía tiene cada cliente en cada depósito de valores al final del día. Si no lo haces en su orden correcto, podrías pagar el dividendo a un cliente al que no debes hacerlo, pues ya no tiene acciones de esa compañía al haberlas vendido ese mismo día. En los tiempos heroicos eran los Operadores los que lanzaban los procesos desde la Consola del Sistema, mirando qué procesos iban acabando, y consultando unos esquemas que ayudaban a saber qué iba antes de qué... Ahora es sencillamente imposible. Ni con un ejército de operadores podría hacerse tal cosa.

Cualquier gran instalación española (o mundial, obviamente) puede lanzar a diario cientos de procesos *para cada Aplicación*: unos son diarios, otros, semanales, mensuales, decenales, bimestrales... Otros son a petición, o sea, es el usuario quien decide cuándo se deben ejecutar; los típicos son los procesos de cierre contable. Otros se deben ejecutar si no sé qué cosa ha pasado en un proceso anterior... Una locura, y cada vez en menos tiempo.

Así que ya no es el sufrido operador quien lanza los procesos: es la propia máquina quien lo hace, gracias a los **Planificadores de Trabajos** (Control/M, de BMC y Tivoli Workload Scheduler –lo que toda la vida se llamaba OPC-, de IBM, son los más usados en el mundo del mainframe), donde se establecen las condiciones de ejecución, prioridades y

dependencias entre los procesos, qué hacer si fallan, etc.

El resultado es que lo normal es ver seiscientos o setecientos procesos batch ejecutándose simultáneamente durante la ventana batch, que suele ser nocturna, como es lógico, con la CPU todo el rato al 100%...

La noche es el periodo más ajetreado de los modernos mainframes, con diferencia... bueno, y de los antiguos, también. Muchas anécdotas hemos vivido durante noches interminables en las que veías cómo iban funcionando tus programas que estabas poniendo en marcha... como debían pasarse por la noche (sí, en la Ventana Batch, naturalmente), no había más remedio que trasnochar si querías estar presente por si había algún error y así poder corregirlo... me da que ahora estas cosas, sencillamente, no se hacen.

Recuerdo una de esas noches compartiendo café y bocadillos con los operadores en que de pronto nos dimos cuenta de que uno de ellos no estaba. Nadie sabía dónde podía estar, le buscamos por todo el Banco, preocupados por él, pues pensábamos que igual le había dado un *yuyu* o que se había tirado por alguna ventana, cuando de repente aparece con cara de “aquí no ha pasado nada...”.

Al final confesó que había ido al baño, se había sentado en la taza... ¡y se había quedado dormido!, que mira que es incómodo el sitio para dormir... nada menos que ¡cuatro horas! La vida del operador nocturno era, sin duda, bastante peculiar.

En este punto de la narración seguro, seguro que, mientras lo leéis, muchos de vosotros estáis pensando: ...Pero, *vamos a ver*, si yo puedo hacer operaciones con mi “Banco en Casa” durante todo el santo día... y toda la santa noche. ...Y puedo comprar artículos en cualquier tienda online a cualquier hora. ...Y puedo usar un Cajero Automático a las tres de la mañana... A ver... *¿qué demonios me está contando este anciano aquí?*

Efectivamente, las cambiantes y siempre acuciantes necesidades de negocio han hecho que **los horarios de atención a los clientes deban ser de veinticuatro horas**, así que **muchas aplicaciones críticas para el negocio han debido evolucionar para adaptarse**. ¿Cómo lo han resuelto las Compañías? De diversas maneras, según el ámbito de actividad.

Es muy normal que a los Bancos en Casa, Cajeros Automáticos, etc., no les atiendan las Bases de Datos “*reales*” durante todo el tiempo, sino sólo mientras esté el online principal, el de las Oficinas, abierto. Cuando se cierra el online se sigue atendiendo *con una copia de la Base de Datos*: para el cliente es la misma cosa... pero las operaciones no se realizan de verdad, se van “apuntando” y al arrancar el online el día siguiente, se consolidan inmediatamente. O sea, para que el cliente no perciba nada raro, las entidades han tenido que montar una buena movida, duplicar bases de datos, aplicaciones, copias de aquí para allá y luego para acullá... ¡todo sea por el cliente!

En las tiendas online es más fácil, aunque también tiene lo suyo: se vende con las existencias del final del día anterior, por lo que puedes vender la misma lata de tomate a dos clientes diferentes, al no actualizar el stock tras cada operación. Pero como alguien tiene físicamente que buscar en los anaqueles la lata de *tomate Albo* que has pedido para meterla en tu paquete, no hay ningún problema. Porque si están agotadas, seguramente alguien te llamará a casa para decirte “*¿Señor Fernández? Mire, que temporalmente no tenemos tomate Albo... ¿Le enviamos Apis, o prefiere anular el pedido del tomate?*” Y Santas Pascuas...

...Y, además, seguro que hay entidades, o determinadas aplicaciones de las

entidades, que **realmente hacen convivir el batch y en online**. Ah, pero... *¿pueden convivir?* La respuesta es **sí**, pero mediante un procedimiento que es muy, muy complicado.

La idea es **gestionar el batch como si se tratase de un conjunto de transacciones individuales** y, por lo tanto, deben emitir una sentencia *COMMIT*, es decir, las sentencias de consolidación de los datos en la Base de datos, con las que avisamos al Gestor: “*Hasta aquí, estoy bien*”, cada  $x$  actualizaciones, siendo  $x$  pequeña, quizá cinco o diez. Hay que tener un cuidado extraordinario con la programación de estos sistemas, asegurándose de que todos los procesos sean *rearrancables (menudo tostón)*, es decir, que si fallan se pueda arrancar de nuevo el proceso por el punto exacto donde se quedó... lo que exige una serie de precauciones y procesos adicionales que además encarecen mucho el consumo de recursos.

Pero sobre todo, exige **tener fe ciega en la calidad de los programas batch** y asumir que si fallan en cierto punto, todo lo que hicieron antes estaba, no obstante, bien hecho. Y eso... eso es mucho asumir, amigos. Claro que, si no queda más remedio... pues no queda más remedio, pero la verdad es que los informáticos huimos de la “*Convivencia Batch-Online*” como de la peste.

Quizá os preguntéis a qué se refiere este viejo carcamal con “**Procesos rearrancables**” y, sobre todo, por qué me refiero a ellos con tanta prevención, casi animosidad...

Si un proceso (o sea, un programa) debe ser rearrancable, debe tener en cuenta no sólo el diseño de sus procesos propios, como liquidar cuentas y todas esas tontadas, sino que tiene que llevar apuntada la información que le permita rearrancar por el punto exacto cuando falla.

Para hacernos una idea, pensemos en un programa más o menos standard de los afectados por esta problemática: Por ejemplo, lee tres ficheros secuenciales y graba dos; accede a diez tablas, de las que actualiza cuatro y genera un listado con las incidencias encontradas. Algunos igual pensáis que me he pasado, que los programas de hoy en día no tienen tanto fichero ni base de datos a que acceder... ¡estáis equivocados, queridos lectores!

Cuando es preciso hacer convivir batch y online, la tendencia es la de **acumular las actualizaciones críticas en cuantos menos programas mejor**, para minimizar los puntos de fallo; los programas resultantes son a veces monstruosos, pero bien estructurados y documentados son perfectamente mantenibles... yo he llegado a ver programas que acceden a cuarenta y pico tablas, de las que actualizan veintitantas, que tienen más de diez mil líneas de código y, sin embargo, son perfectamente comprensibles y mantenibles. De veras.

Sigamos. Tenemos nuestro hermoso programa batch (suponemos que lo hemos probado exhaustivamente y funciona) que tiene que convivir con el online, o sea, ejecutarse mientras el online está abierto, así que pueden producirse actualizaciones provenientes de transacciones en cualquier punto y momento. Y tenemos que preparar el programa para que, en caso de fallo por el motivo que sea, pueda rearrancar y continuar el proceso exactamente por el punto donde se quedó.

Veamos algo más en detalle los puntos que hay que tener en cuenta:

- 1 Además de los propios ficheros y bases de datos que use para su proceso primario, **hay que definir unos ficheros especiales para el control del rearranque**. Mejor fichero que base de datos, pero también podría ser. En este fichero hay que guardar toda la información de por dónde íbamos, que nos permita rearrancar; en cada caso será

distinta, puede bastar con la clave del registro por el que vamos (por ejemplo, la cuenta), o añadir alguna información adicional, según las necesidades concretas. Este fichero debe ser meticulosamente borrado antes de la ejecución del programa de marras (de *la primera ejecución*, quiero decir).

**2 La estructura del programa es muy distinta a la que tendría sin “rearrancabilidad”.** En primer lugar, el programa debe leer su fichero de control de rearranque, para ver cuál es la situación de su ejecución. Si está vacío, significa que tiene que comenzar por el principio. Pero si tiene contenido, significa que ya ha procesado parte de la información en un intento anterior y tiene que seguir por donde iba. Así que tiene que posicionar todos los ficheros y bases de datos en el mismo punto donde cascó la llamada anterior... Habrá que leer ficheros hasta que la clave coincida con (o sea superior a, según el diseño que se haga) la que se encuentra en el fichero de control. A las Bases de Datos habrá que hacerles lecturas para dejar cursores o posicionamientos en el mismo punto exacto en que se quedó cuando falló... En fin, que se trata de un proceso complejo, laborioso y en el que hay que ser muy cuidadoso para no llevarse sorpresas.

**3** Una vez *posicionados* en el lugar correcto de los datos, bien por estar al principio de todo, bien mediante el proceso de reposicionamiento anterior, hay que empezar el proceso *de verdad*, pues hasta ahora no eran más que *juegos florales*. **Pero también la estructura de la parte mollar del programa debe sufrir cambios importantes:** ahora, cada pocos registros (cuentas, movimientos, etc., depende de cada programa) se debe emitir una sentencia de consolidación de las Bases de Datos: un COMMIT. Y se debe, además, escribir en el fichero de control por dónde vamos. Es decir, abrir el fichero, escribir el registro de control, y *cerrarlo*, para consolidarlo igual que las bases de datos. He dicho “cada pocos registros” (quizá 5 ó 10), aunque también se puede hacer a cada uno de ellos, pero el propio proceso de COMMIT consume bastantes recursos, así que normalmente se llega a un compromiso. Salvar cada diez registros, nos garantiza que, en caso de fallo, sólo hemos perdido el trabajo de como mucho los últimos nueve, y a cambio hemos reducido mucho el tiempo de proceso.

**4** Naturalmente, **hay que elegir muy bien el punto exacto donde se realiza la consolidación**, para no dejar nada a medias, es decir, hay que convertir el proceso batch en un *proceso transaccional por lotes*...

No sé si ha quedado claro, pero incluir la capacidad de rearrancar en un programa es un tostón para diseñador y programador. Y no creáis que esto de la rearrancabilidad se inventó con las Bases de Datos, no... Yo ya hacía programas rearrancables para el venerable NCR Century 200, donde no había ni bases de datos, ni online, ni nada... esta vez era por motivos muy distintos: ¡**ahorrar papel!**

Vale, ya me explico: tú tenías un magnífico programa que imprimía el Balance de Cuentas Corrientes, sin ir más lejos... un tocho de papel pijama de ochenta centímetros de altura y cuarenta o cincuenta kilos de peso con todas las posiciones de las Cuentas Corrientes de todas las Oficinas.

Aquel programa hacía impresión directa: estamos en un sistema monoprogramación y no existe aún SPOOL (que no es un nombre de nada, por cierto, como la mayoría de informáticos cree: SPOOL es el acrónimo de “*Simultaneous Peripheral Operating On*

*Line*”). O sea, que cuando el programa dice “WRITE LINEA”, esa línea se imprime físicamente en ese mismo momento por la impresora... siempre que ésta esté *ready*, que si no, se espera hasta que lo esté. Sin más.



Todo va como la seda... y de pronto **¡se rompe el papel continuo!** (o se va la luz, o el operador se equivoca y aprieta inopinadamente la tecla de Stop... cualquiera de esas causas tan habituales en aquellos lejanos años). Y ya llevaba impresos sesenta y cinco de los ochenta centímetros de papel...

Volver a empezar el listado por el principio significa tener que enviar a destruir kilos y kilos de papel que está ya impreso, y encima con la información correcta, que además es confidencial. No parece muy lógico destruirlo, ni desde el punto de vista económico, ni desde el ecológico, aunque en realidad, en los años setenta, yo creo que ni siquiera sabíamos que existiera la palabra *ecología*.

Entonces el operador buscaba en el papel la última cuenta que había quedado bien impresa, la introducía al arrancar el programa de nuevo y ahora el programa leía registros hasta colocarse en ese punto exacto y comenzaba a imprimir por donde se quedó la vez anterior... A grandes problemas, ¡grandes soluciones! Pero aseguro, y lo sé bien, que *era un tostón* escribir programas así.

Y hasta aquí el **proceloso mundo del batch y sus ventanitas...**

## 13 – Aparición fulgurante de las Metodologías de Desarrollo

Tras los capítulos dedicados a la aparición de los PC's como tecnología viable, a las Bases de Datos, sobre todo Relacionales, y a los procesos batch generalmente concentrados, por no decir *apelotonados* en la famosa Ventana Batch, este capítulo versa sobre la tercera tecnología de gran calado que apareció en (y cobró enorme importancia durante) la segunda mitad de los ochenta: Las **Metodologías de Desarrollo**. Al poco comenzaron a aparecer como setas en un día soleado de otoño decenas de Herramientas Informáticas para facilitar la cumplimentación de las técnicas incluidas en ellas: el software para hacer Ingeniería de Software Asistida por Ordenador, es decir, las **Herramientas CASE** (CASE Tools, según su nombre en inglés), aunque en ellas me centraré en el próximo capítulo.

Repito lo mismo de siempre: no es mi intención escribir una crónica oficial de nada, sino más bien cuáles son mis recuerdos de las vivencias de aquella época, la segunda mitad de los ochenta. El advenimiento de las Metodologías y de las Herramientas CASE lo viví también muy de cerca... igual diréis que cómo es posible que yo estuviera metido en tantos *fregados tecnológicos* al mismo tiempo... en estos tiempos modernos es muy difícil para un técnico, por muy listo que sea, estar al tanto de todo lo que acontece en cada parcela de la técnica informática. Entonces tampoco era posible estar al tanto de todo avance... pero sí de la mayoría, siempre que se fuera muy, muy curioso y se tuviera la suerte de estar trabajando en el puesto adecuado de la empresa adecuada... lo que, afortunadamente, era mi caso durante los ochenta y primeros noventa del siglo pasado.

Ya comenté hace un par de capítulos que la aparición de las Bases de Datos Relacionales había significado un shock para la profesión: implicaban la utilización de técnicas de diseño de Base de Datos bastante diferentes de lo que estábamos acostumbrados, mucho menos “personalistas”, si se me permite la expresión, y mucho más industriales. Es decir, comenzaba a vislumbrarse el mismo camino que pocos años antes habían seguido las técnicas de programación, con la aparición y rápida adopción de la Programación Estructurada.

Por tanto, el sentimiento de los profesionales en la época, mediados de los ochenta, y entre ellos el mío propio, era algo así como: “*Vaya, por fin comienza a ser posible unificar formas y métodos y encontrar un lenguaje común que nos permita una comunicación precisa entre nosotros...*”. ¡Cuán equivocados estábamos...! Pero como entonces no lo sabíamos hicimos grandes esfuerzos para lograr ese “*idioma común*”.

Y todo, porque la situación estaba cambiando a pasos agigantados. Los últimos años de los ochenta y primeros de los noventa fueron de un enorme crecimiento en todos los aspectos de la informática: crecimiento en potencia y variedad de las máquinas, en Sistemas instalados, en Aplicaciones funcionando (ya, casi todas, online) y, sobre todo, de **fuerte crecimiento del personal** de los Departamentos de Informática, o Proceso de Datos, como seguían llamándose mayoritariamente por entonces: no se había consolidado aún la moda del marketing a los nombres de los Departamentos dedicados a estas cosas.

Casi podría asegurar que, en el plazo de tan sólo tres o cuatro años, el plantel de informáticos al menos se duplicó en la mayoría de grandes empresas, y en algunas bastante



más. Y no fue nada fácil encontrar a tanto informático. Doy fe. ¡La de currícula que me leí y la de entrevistas que pude hacer esos años!

Costó un gran esfuerzo poder incorporar tanto técnico informático a las empresas, simplemente porque en el mercado no los había. Todas las Universidades Españolas juntas quizá podían graduar a, digamos, mil informáticos anuales, seguramente bastantes menos... lo que era totalmente insuficiente para crecimientos como los que entonces se estaban dando en España.

Como consecuencia, no hubo más remedio que echar las redes en los caladeros de cualquier otra especialidad, sobre todo las técnicas, lo que entonces se conocía como “*Carreras de Ciencias*”: Matemáticas, Física, Biología, Geología, Ingenierías, sobre todo “Teleco” (aunque menos), incluso en ¡Medicina! Éstas eran carreras que proporcionaban una buena preparación de base, algún conocimiento de informática y que, con la excepción de Medicina, no tenían mucha salida en España quitando la docencia, que obviamente pagaba bastante peor que la informática... y que, además, por entonces tenía menos *glamour*, qué demonios (ahora, de ninguna manera, pero ésa es otra historia y será contada en otro momento).

Así que las grandes empresas nos lanzamos a contratar Licenciados, Diplomados, Peritos o lo que encontráramos (sí, también algún que otro *recomendado*, por si lo os lo estáis preguntando, pero menos de los que cabría esperar) y a formarlos en la tecnología pertinente, mayormente Cobol, DB2, IMS o CICS, según el caso, y toda la parafernalia *mainframera* asociada: ISPF, JCL, SPUFI, etc. Y Programación Estructurada, naturalmente. Porque a mediados y finales de los ochenta, todas estas cosas mundanas ya casi no se explicaban en las Universidades españolas (podéis imaginaros cómo es ahora), a pesar de que eran las que daban trabajo a quizás siete u ocho de cada diez informáticos de los de entonces.

Todo este movimiento trajo como consecuencia un efecto colateral, una *desmitificación de la profesión*. Porque, total, si con un curso de un par de meses o tres convertíamos a un biólogo, un químico experimental o un físico de partículas en un informático y lo poníamos a programar, y luego no lo hacía tan mal... ***no sería esto tan difícil***, ¿no?

Los Departamentos de Recursos Humanos, asustados ante la necesidad de contratar miríadas de informáticos de elevado sueldo, a los que encima había que formar a precio de oro antes de que fueran capaces de escribir su primera MOVE, comenzaron a ajustar los salarios. A reducirlos, a optimizarlos, vaya. A *racanear*. Dejaron de creerse, ya para siempre, el mantra que repetíamos continuamente de que “*los informáticos somos unos sofisticados técnicos, conocedores de un extraño e ininteligible arcano, y que, por tanto, rara avis como somos, debemos tener un (enorme) sueldo en consonancia*”. Allí fue donde comenzó nuestro declive, amigos. Y así seguimos...

Bueno, y... ¿Por qué cuento yo todo esto?

Pues el motivo es porque, con tanta incorporación de personal nuevo comenzaba a ser evidente que era necesario llegar bastante más allá de una simple *recomendación de uso* de ciertas técnicas: **había que asegurarse**, en lo posible, de que tanto **el procedimiento de Diseño de las Aplicaciones** como **la Documentación asociada** estuvieran **formalizados**, uniformizados, reglamentados, constreñidos de tal modo que cualquier persona pudiera leer, comprender y, llegado el caso, modificar cualquier Aplicación sin necesidad de apelar constantemente a la memoria o al conocimiento de su creador...

Tenía yo un compañero, algunos años antes de todo esto, andaluz y muy gracioso él, que se autodefinía como “*El Hombre-Aplicación*” porque llevaba la Aplicación de Nómina en la cabeza igual que los “*Hombres-Libro*” de la novela “*Fahrenheit 451*”, de Ray Bradbury, se aprendían de memoria un cierto libro, como Hamlet o El Quijote, para salvaguardarlo de la malsana campaña de quema de libros promovida por el gobierno, sólo que en este caso, si al pobre hombre le daba por irse de vacaciones o enfermar... ¡igual no se podía pagar la Nómina ese mes!

Ésta era una situación que se daba con alguna frecuencia y que las empresas no estaban dispuestas a que volviera a ocurrir en el futuro. Aunque igual sabéis que, en realidad, por mucho que documentos y que mantengas la documentación, sigue siendo mucho más eficaz que realice cualquier cambio a la Aplicación alguien que la conozca que alguien que no sepa de qué va, por mucha documentación electrónica o en papel que haya. No obstante, siempre será mejor que exista documentación a que no. Y si es buena, mejor. Y encima, si está actualizada... bueno, no. Actualizada, lo que se dice actualizada del todo no lo está nunca, ya lo sabéis.

Parecía, pues, evidente que había que implantar un procedimiento de Diseño, Construcción y Documentación de Aplicaciones que permitiera eliminar a los “*Hombres-Aplicación*” (o a las “*Mujeres-Aplicación*”, que también había).

Para ello se necesitaba, en primer lugar, un **Método** que dijera qué es lo que había que hacer y qué no; en segundo lugar, unas **Técnicas** de uso común para realizar el Análisis, el Diseño y la Programación, aunque ciertamente esta última ya estaba bastante formalizada por aquellos años; y en tercero, a ser posible, un **Sistema informático** de algún tipo que se asegurara de que las Normas del Método se cumplen y las Técnicas se aplican.

Estas necesidades básicas son solucionadas por las **Metodologías de Desarrollo de Software**, por las **Técnicas de Análisis y Diseño** y por las **Herramientas CASE**, respectivamente. Pero... ¿cuál era el estado de cada una de ellas a mediados de los ochenta? Allí vamos, aunque repito que en este capítulo hablaré, sobre todo, de Metodologías y de Técnicas; en las Herramientas CASE, siglas de *Computer Aided Software Engineering*, me centraré en el próximo.

Sobre las Metodologías... pues, en la práctica, no había Metodologías de Desarrollo tal como las conocemos ahora. Sí que había algunas, de las que hablaré brevemente en un momento, que eran más bien un listado de tareas y documentos que había que cumplimentar, firmar y guardar, sobre todo para guardarte las espaldas cuando las cosas no fueran como se suponía que debían ir... Y esto porque, en su mayoría, fueron promovidas por ciertos Gobiernos para asegurar el cumplimiento de las especificaciones y, sobre todo, de *los costes* de los grandes proyectos contratados por la Administración.

En cuanto a Técnicas, hemos visto cómo la Programación Estructurada había triunfado ya, pero apenas había nada sobre cómo realizar el Diseño Técnico, ni mucho menos el Análisis Funcional. Para el Diseño Técnico ya se escribían y guardaban con cierto formato los Cuadernos de Carga de los programas, pero a la hora de explicar lo que debía hacer el programa, se explicaba... en cristiano... como se le ocurría al Analista Orgánico. Sin ir más lejos: “*Abrimos el fichero de entrada; leemos registros y consultamos en la Base de Datos BDCUENTA si la cuenta existe; si existe, patatín, patatán... si no existe, patatán, patatín...*”, y así.

Y en cuanto al Análisis Funcional, se daban reglas, sugerencias... pero el Analista, en definitiva, escribía su Análisis Funcional explicando en román paladino lo que debía hacer la Aplicación como buenamente podía, sabía y se le ocurría.

Luego, en cuanto al diseño de las Bases de Datos, tampoco había mucho más que la experiencia del Analista y las Reglas de Normalización de Codd, siempre que las Bases de Datos fueran Relacionales, porque si eran jerárquicas ni siquiera eso.

Y por fin, en cuanto a las Herramientas informáticas, alguna había, pero muy orientadas a facilitar el trabajo del programador, que era, en verdad, el único que trabajaba con herramientas informáticas de verdad: el TSO, el ISPF, etc., pues el resto de roles del Desarrollo de Aplicaciones trabajaban mayormente en aquellos años con un lápiz y un papel... y con gasto de saliva, sobre todo, mucha saliva.

Quizá conviene resaltar aquí que ya entonces el Mantenimiento de Aplicaciones era un serio problema... y tenía toda la pinta de convertirse rápidamente en **El Problema**.

En la imagen siguiente tenemos un gráfico más o menos prototípico de los costes de cualquier gran Aplicación a lo largo del tiempo.



La Concepción del Sistema, el Análisis Funcional, el Diseño Técnico, el Desarrollo, todas las Pruebas, etc., es decir, todo el coste incurrido hasta la puesta en marcha de la Aplicación es la zona verde claro de la curva.

Una vez puesta en marcha comienza su fase de mantenimiento, la zona naranja oscura, que supondrá seguramente, a lo largo de toda la Vida de la Aplicación, entre un 75% y un 90% del coste total dedicado a dicha Aplicación a lo largo de su vida útil. Nosotros llamábamos a este gráfico “la *ballena*”, supongo que es obvio el por qué...

Además, las proyecciones eran palmarias: de seguir así, *en unos pocos años el 90% de la plantilla de Desarrollo o más estaría dedicada exclusivamente al Mantenimiento...* y entonces, ¿quién escribiría las nuevas Aplicaciones? Nuevo personal, claro está. Que habría que fichar o subcontratar, obviamente. Y que, una vez terminada su Aplicación, quedaría, en un 90%, dedicado a su mantenimiento. Y habría que fichar entonces más personal nuevo para escribir las nuevas aplicaciones...

No parecía que el círculo vicioso originado por esta forma de trabajar pudiera resultar muy eficiente a la larga. Así que las connotaciones son evidentes: **No hay que escatimar esfuerzos durante la etapa inicial de Diseño y Construcción de la Aplicación para reducir todo lo posible el coste del Mantenimiento posterior.** Es decir, incrementar un 20%, un 30% o incluso un 50% el coste del Desarrollo de la Aplicación no supondrá, a

la larga, un porcentaje importante del coste total de la Aplicación durante toda su vida útil, quizás un 5%-10% del coste total, mientras que reducir un 20% o un 30% el coste del Mantenimiento tendrá a la larga un efecto muy importante en los costes finales acumulados.

Fue en aquellos años cuando empezamos a escuchar en Congresos (los pocos que íbamos) y leer en revistas (los que las leíamos, que ya éramos algunos más, siempre que no estuvieran escritas en inglés, claro) sobre el “*nuevo paradigma*” de la **generación automática de código**. Qué bonito sonaba, ¿no?

La idea es que, una vez introducidas las especificaciones de la Aplicación en el Sistema, descritas formalmente en algún tipo de idílico lenguaje por inventar, el supuesto Sistema sería capaz de generar mágicamente todo el código necesario para que la Aplicación funcione correctamente: las definiciones de las Bases de Datos, con sus índices, vistas y toda la parafernalia, la definición de las pantallas de interfaz, la codificación de los programas, los JCL's de las cadenas batch... ¡todo!, es decir, que apretando un botón se obtendría milagrosamente todo el código necesario. Lo de “*apretar un botón*” y que todo lo que se desea salga automáticamente -mejor: *milagrosamente*- es una vieja aspiración de todo informático y, sobre todo, de todo usuario: un jefe que yo tuve decía que había que dejarse de zarandajas y que había que inventar de una vez por todas la “*Máquina-con-Orejas*”... y que sea *obediente*, añadía yo cuando le oía, porque como nos saliera peleona...

Había ya algunas Herramientas aquellos años (mediados de los ochenta, para que no os perdáis) que eran capaces de generar cierto código en base a ciertas especificaciones. Las que más se conocían en España eran **Pacbase**, de la francesa CGI (a mediados de los noventa, CGI fue comprada por IBM y, como consecuencia, Pacbase fue incorporada, ignoro con qué éxito, dentro de ese galimatías de productos llamado VisualAge), y **MAESTRO**, de la alemana Softlab, aunque en España la vendía Philips. Cuando a fines de los ochenta apareció su sucesor, llamado, con gran esfuerzo imaginativo por parte del departamento de marketing de Softlab, “*Maestro II*”, este MAESTRO-*a-secas* primigenio pasó a denominarse Maestro I.

Ninguna de las dos podría considerarse aún como una auténtica “Herramienta CASE” como las que conocemos hoy en día, sino más bien como un *Producto que ayudaba al programador*, básicamente, a *hacer su trabajo más rápido*.

Mientras Pacbase eran sólo unos programas, un software que funcionaba en las propias máquinas para las que generaba su código, es decir, en el propio mainframe bajo TSO, MAESTRO era un conjunto de hardware (un miniordenador de Motorola vendido en Europa por Philips, como “*Philips P7000*”) y un software muy bien hecho, el propio *Maestro* en sí, que corría en este miniordenador.

En la ilustración siguiente se puede observar una foto del ordenador Philips P7000, así como del teclado específico para Maestro.



A Pacbase se podían conectar todos los usuarios que fuera capaz de soportar el TSO, o los que se tuvieran contratados, pero cada MAESTRO, es decir, cada máquina, sólo podía dar servicio como máximo a un cierto número de usuarios (16, 24 o quizá 32, no

recuerdo bien), que usaban pantallas “tontas” conectadas por cable coaxial con la CPU. Por el diseño de la máquina utilizada, inicialmente pensada como máquina de Data Entry, era rapidísima, muy similar en tiempo de respuesta a los de un PC actual: cuando pulsabas una tecla, por ejemplo, la A, esa A aparecía inmediatamente en tu pantalla (de 24 columnas de 80 caracteres, naturalmente) y se grababa en el disco.

Lo que editabas y guardabas eran, fundamentalmente, programas que luego debías enviar al mainframe para su compilación y prueba. Para ello se conectaba a los mainframes de IBM (supongo que también a los de otras marcas) de dos formas: por “emulación 3270” para conectarse al TSO, al CICS, etc., o por “RJE” para submitir trabajos batch para su ejecución en el mainframe: compilaciones, pruebas, etc.

El gran argumento de marketing de Philips-Softlab era precisamente la velocidad: editar en MAESTRO un programa, una documentación, un JCL o lo que fuera era tan rápido como puede serlo hoy en día... sólo que lo normal entonces era que en TSO los tiempos de respuesta fueran de varios segundos, de media ocho o diez, para cada interacción, por ejemplo, un avance de página, insertar una línea, realizar la búsqueda de un texto, etc. Este argumento tenía un hermoso corolario: reducir o eliminar usuarios de TSO del mainframe, usuarios caros y, sobre todo, muy voraces en consumo de CPU permitía demorar el inevitable crecimiento vegetativo del ordenador central. Este ahorro de costes de mainframe en ocasiones ya compensaba por sí solo la adquisición de un sistema como Maestro, que no era nada barato, creo recordar.

Así vendió Softlab muchos sistemas en todo el mundo: MAESTRO fue el líder absoluto en puestos de trabajo de desarrollo durante unos años, a finales de los ochenta: llegó a tener varias decenas de miles de puestos de trabajo operativos en todo el mundo.

Pero las cosas iban a cambiar rápidamente, para desgracia de Softlab... bueno, y de CGI, que con su Pacbase dominaba el mercado francés de largo aunque, que yo sepa, nunca vendió mucho fuera de Francia.

En España ambos productos tuvieron un éxito limitado, consiguiendo algunas ventas en empresas muy importantes... pero pocas. Claro que tampoco había tantas empresas *importantes* en España, y para poder siquiera considerar la compra de estos productos, que eran realmente caros, había que tener una masa crítica considerable que permitiera amortizarlos en un plazo razonable.

Primero, porque los PC's comenzaban a ser asequibles, al menos *más* asequibles que en los años precedentes y todo el mundo estaba asumiendo que todo técnico iba a acabar con uno encima de la mesa más temprano que tarde. Y segundo, porque varios ingenieros de la computación estaban atacando ya (de hecho, llevaban algunos años en ello) el problema siguiente al de cómo realizar eficientemente la Programación, es decir: **¿cómo realizar el Diseño de los Sistemas?**

Es decir, realizar de alguna manera más formal que simplemente escribir la documentación en cristiano, aunque fuera en un procesador de textos en vez de con boli, tanto el Diseño Técnico del Sistema: Bases de Datos, Transacciones, Procesos Batch, etc., como el propio Análisis Funcional.

Uno de los esfuerzos vino del propio Michael Jackson, el inventor del método de estructuración de programas que lleva su nombre, que en 1983 publicó su libro “System Development”, en el que presenta su método JSD, aunque ya estaba dando cursos y seminarios sobre el asunto desde hacía un par de años.

En realidad JSD es una extensión de su método de Estructuración de Programas, esta vez orientado al Diseño de los Sistemas. No tuvo excesivo éxito, pues llegó algo tarde,

pero fue nuevamente Jackson quien hizo una formalización extraordinaria: su método fue el primero que estableció que **los Sistemas deben ser diseñados a partir de las Salidas**, es decir, de los resultados esperados. Eso que ahora es tan evidente lo estableció él... ¿o... *quizá no es tan evidente?*

Bueno, para mí sí que lo es. Todos los procesos, los datos necesarios, todo es la consecuencia lógica de conocer para qué debe servir nuestro sistema, es decir, qué es lo que esperamos que haga en cada caso, lo que debe de producir. En definitiva: las salidas. Malamente vamos a escribir un Sistema que no sabemos qué información es la que tiene que obtener...

Efectivamente, el método de diseño de Jackson llegó algo tarde, pues algún tiempo antes otros importantes *gurús* de la informática habían ya publicado diferentes visiones para sistematizar el proceso de Análisis y Diseño Estructurado de Sistemas.

Peter Chen había atacado con éxito la problemática de la modelización de los datos mediante su técnica “**Modelo de Entidad-Relación**” (ERM), publicado inicialmente tan pronto como en 1976, pero que no tuvo aceptación real hasta que las Bases de Datos Relacionales comenzaron a invadir las instalaciones de todo el mundo, a mediados y fines de los ochenta. El motivo es obvio, ya que, una vez realizado el Modelo de Datos según su método, es prácticamente inmediato obtener una estructura de Tablas Relacionales, estructura que no tiene por qué ser muy eficiente o, mejor aún, que rara vez será eficiente pero, en principio, sí es factible implementarlo.

Esta forma de modelizar los datos definida por Chen fue adoptada por virtualmente todos los métodos de Desarrollo de Aplicaciones de entonces (y de ahora), debido a su consistencia y, sobre todo, a su facilidad de aprendizaje y comprensión.

Utiliza apenas tres elementos bastante intuitivos: **Atributos**, es decir, los diferentes datos individuales que se precisan para nuestro Sistema de Información, **Entidades**, que son los contenedores de los atributos, que deben estar una y sólo una vez en cada ocurrencia de la Entidad, y **Relaciones** entre Entidades, que modelizan qué tipo de conexión hay entre dos diferentes Entidades.

El nombre de la técnica: **Modelo de Entidad-Relación**, describe perfectamente sus constituyentes fundamentales.

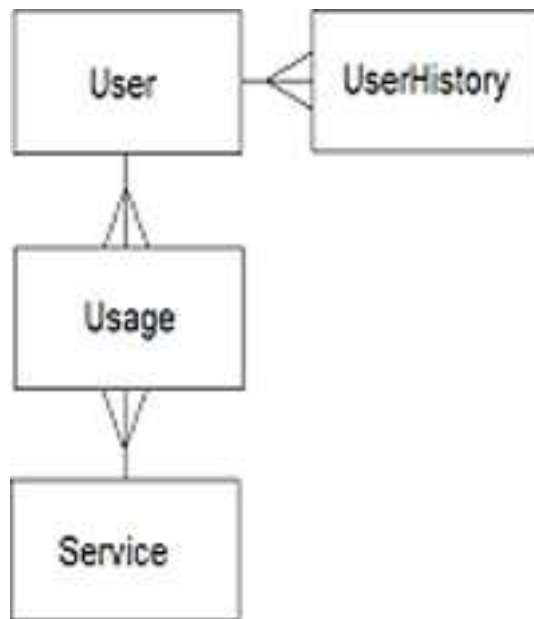
Las Relaciones que indican las correspondencias entre dos Entidades pueden ser de tres tipos (supongamos en lo que sigue una relación entre las Entidades A y B):

**a) de uno a uno.** Por cada ocurrencia de la Entidad A hay una y sólo una ocurrencia en la B; se representa  $1:1$ ;

**b) de uno a muchos.** Para cada ocurrencia de la Entidad A hay un número indeterminado de ocurrencias relacionadas en la Entidad B, pero cada ocurrencia de la Entidad B sólo se relaciona con una única ocurrencia de la Entidad A; se representa  $1:n$ ;

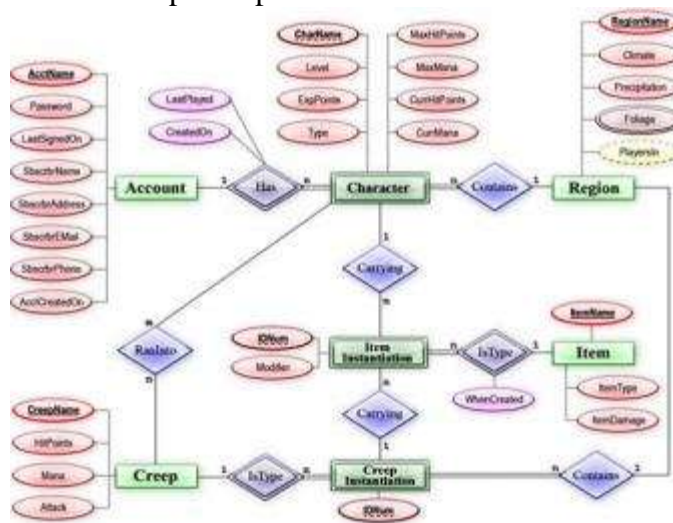
**c) de muchos a muchos.** Para cada ocurrencia de la Entidad A hay un número indeterminado de ocurrencias relacionadas en la Entidad B, mientras que para cada ocurrencia de la Entidad B hay también un número indeterminado de ocurrencias relacionadas en la Entidad A; se representa  $n:m$ , y para su implementación física debe antes ser descompuesta mediante una Entidad intermedia con los datos de intersección, y dos relaciones  $1:n$  en la Fase de Diseño Técnico, es decir, para definir las tablas Relacionales definitivas.

En el ejemplo siguiente, la entidad “Usage” es la que descompone la Relación  $n:m$  que hay entre “User” y “Service”.



Este tipo de nomenclatura se refiere al *número máximo* de elementos de una entidad que están relacionados con otra; sin embargo, para realizar la definición formal completa hay que especificar también el *número mínimo* de elementos de la relación.

Por ejemplo, en una relación  $1:n$ , si es posible que cada entidad pueda estar relacionada con *de cero a ene* elementos de la otra, habría que especificarlo de la forma  $1:(0,n)$ . Lo que ocurre es que en la vida real es muy difícil encontrar relaciones del tipo  $1:(1,n)$ , por ejemplo. Lo normal es que sean todas  $1:(0,n)$ , por lo que, si no se dice nada en contra, se asume que la  $n$  (o la  $m$ , claro) puede valer cero también, es decir, el valor *cero* es también un valor aceptable para el término *muchos*.



En la imagen tenemos un Modelo de Entidad Relación, a saber de qué extraño Sistema, cortesía de la Wikipedia.

Veamos unos pequeños ejemplos de Entidades Financieras, que ya sabéis que son las que mejor me sé:

**a) Oficina – Dirección de la Oficina** (*de uno a uno*): Una Oficina está en un único domicilio (Dirección), mientras que en cada Dirección hay una única Oficina: no

tiene lógica mantener dos Oficinas diferentes de la misma entidad en la misma ubicación. Bueno, *para mí* no tiene ningún sentido, pero cosas más raras se han visto.

**b) Cuenta – Movimiento** (*de uno a muchos*): Cada Cuenta puede tener muchos movimientos a lo largo de un periodo, un número en principio indeterminado (desde ninguno a cientos, o miles), pero cada apunte pertenece a una única Cuenta. En el caso de la típica Operación que involucra dos Cuentas, un traspaso de fondos, por ejemplo, se generan dos movimientos diferentes, un adeudo en la cuenta que emite los fondos y un abono en la que los recibe, por lo que cada apunte es de una y sólo una cuenta.

**c) Cliente – Cuenta** (*de muchos a muchos*): Cada Cliente puede ser titular de varias Cuentas, mientras que cada Cuenta puede tener varios titulares. Fijaos en este caso que, en principio, podría haber Clientes sin Cuentas asociadas, pero toda Cuenta debe estar necesariamente asociada a un Cliente; la Relación Cliente-Cuenta debe, por tanto, especificarse como  $(1,n):(0,m)$  para ser formalmente correctos.

Esta técnica de modelización fue rápidamente adoptada por casi todo el mundo, como ya dije, y también en España; de hecho sigue siendo la técnica para modelizar datos utilizada por prácticamente todas las metodologías al uso para realizar el Modelo de Datos.

Sin embargo, había otros gurús que pensaban justo lo contrario que Jackson y Chen pensaban: según ellos, **lo importante de las aplicaciones no son los datos, sino los Procesos**, es decir, conocido lo que hay que hacer, los datos necesarios para hacerlo se hacen evidentes, pues son sólo el subproducto de los procesos. Para mi gusto esto es mucho, pero que mucho decir, pero el caso es que Edward Yourdon y Tom de Marco propusieron como solución a estos avatares su Técnica de Análisis y Diseño Estructurado.

Se basa en una aproximación “top-down” al problema del Diseño de los Sistemas de Información, es decir, comenzar con el diagrama más general e ir luego descomponiendo sistemáticamente el Sistema desde lo más general a lo más particular. Ciertamente es ésta una técnica muy utilizada en casi todos los ámbitos de la ciencia y la tecnología... ¿Por qué no iba a servir en informática?

El resultado de la aplicación de la técnica son los Diagramas de Flujo de Datos (los DFD's), que utiliza exclusivamente cuatro símbolos: Uno para representar los **Procesos** (o Funciones del Sistema), otro, para los **Almacenes de Datos** del Sistema (ficheros o Bases de Datos: en inglés *Data Store*; no confundir con *Data Warehouse*, concepto que se definió años más tarde, y que sirve para otras cosas, pero ésa es otra historia y será contada en otro momento), otro para las **Entidades Externas** (Usuarios u Otros Sistemas, que son los que proporcionan o reciben la información) y el último, para representar los **Flujos de Información** entre los Procesos y los Almacenes de Datos o Entidades Externas, es decir, los datos que el Proceso necesita o genera.





En la imagen están representados los cuatro símbolos utilizados en la definición de los Diagramas de Flujos de Datos; combinándolos entre sí según sus normas de diseño, se obtienen los DFD's definitivos.

El primero de los que es preciso realizar es el Diagrama de más alto nivel; posteriormente, cada Proceso de Alto Nivel se va definiendo mediante un nuevo DFD de segundo nivel, cuyos procesos son a su vez definidos con más DFD's de tercer nivel... hasta llegar al mínimo nivel de definición deseado, que puede ser tan mínimo como queramos; generalmente se queda al nivel del programa individual.

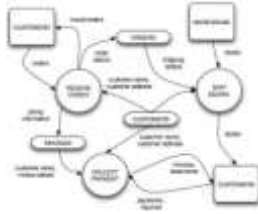
Con razón se llaman “*Métodos Estructurados*”; si se listaban todos los diagramas juntos resultaba un precioso árbol de procesos que se iban descomponiendo a su vez en procesos de menor nivel... Muy *estructurado*, desde luego.

Uno de los personajes más importantes de la informática del siglo pasado, el británico James Martin, se adhirió pronta y entusiásticamente a las técnicas de modelización y creó o quizá sólo adoptó, pero en cualquier caso fue el que dio a conocer, en 1981, la Metodología **IE** (*Information Engineering*), que tuvo inmediatamente una muy buena aceptación... en el mundo anglosajón; en España, muy poca.

Pero, eso sí, todos fuimos a seminarios, leímos artículos, en fin, nos *culturizamos* en IE. Combina una lista de tareas y actividades a realizar para llevar a buen término los proyectos informáticos con el uso de las dos técnicas de modelización antes mencionadas: la de procesos, vía los DFD's, y la de datos, vía los ERD's. James Martin se convirtió en el máximo defensor y valedor de las nuevas técnicas y de los nuevos métodos; buena parte del éxito de ambos en los ochenta y primeros noventa es debido a él.

Adoptar una Metodología de Desarrollo de Aplicaciones trata, por un lado, de minimizar los riesgos del proyecto, controlar costes, asegurar la funcionalidad prevista, gestionar los tiempos, facilitar el mantenimiento... en fin, *todas esas cosas que nos encanta citar a los informáticos*, aun a sabiendas de que no hay manera, por mucha *lujosa metodología* que se use, de asegurarlas. Y también, cómo no, de utilizar un Lenguaje Gráfico de Diseño fácilmente comprensible, no sólo por los propios informáticos, lo que ya era importante *per se*, sino también por Usuarios finales, *por los paganos* del Sistema, cosa que igualmente sabéis que no es tan cierta como aseguramos... pero así andamos, que hay que vender proyectos o desaparecemos del mapa.

En la imagen, un DFD típico, extraído también de la Wikipedia



A partir de aquí, en la meca de la informática, EEUU, se comienzan a utilizar las metodologías de desarrollo, primero de forma tímida, luego con cada vez mayor aceptación... o, al menos, eso nos decían.

¿Y en Europa? Pues también se estaba trabajando en todos estos aspectos metodológicos, pero con curiosas connotaciones nacionales.

En Francia existía desde hacía unos años (fines de los setenta) la metodología **Merise**, en principio una relación (exhaustiva, eso sí) de pasos y tareas con alguna técnica manual sencilla, que también a principios de los ochenta se enriqueció con las nuevas técnicas de modelización de Datos y Procesos. Se trata de un método francés, pensado por y para franceses, que tuvo pocos seguidores fuera del mundo francófono, pero como la Administración francesa exigió el uso de Merise para la contratación de proyectos informáticos desde muy pronto (principios de los ochenta), se convirtió en el standard *de facto* en todo el mercado francés al final de la década de los ochenta.

En el Reino Unido y en la misma época se estaba gestando **SSADM**, que adoptó entre sus técnicas una mezcla de varios de los incipientes métodos de análisis estructurado del momento, en particular los de Yourdon, De Marco, Jackson y Constantine, que fueron incorporados desde el principio al método, se supone que tomando lo mejor de cada uno de ellos. A partir de 1983 también fue obligatorio el uso de SSADM para contratar proyectos para la Administración británica, por lo que fue ampliamente seguida en el Reino Unido y después, supongo que por la cosa del idioma, también en el resto de Europa.

¿Y en España? También hacíamos nuestros pinitos, no os creáis. A mediados de los ochenta el Ministerio de Administraciones Públicas español comenzó el desarrollo de una Metodología española a imagen y semejanza de la francesa y la británica, pero ¡mejor!, que recibió el nombre de “**Métrica**”. Participaron diferentes Organismos de la Administración y se encargó su realización final a la consultora Coopers & Lybrand, que por entonces no estaba aún fusionada con Price Waterhouse, ni mucho menos con IBM, como ahora lo está.

Ahora toca un poco de *queja habitual*: como de costumbre con lo que ocurre en España, se encuentra muy poca documentación de Métrica en la red, y la poca que hay es de la versión 3, ya de fines de los noventa...

Métrica se parecía, inicialmente, a SSADM, pero se le aligeró de bastantes pasos burocráticos para adaptarlo más a *nuestra mentalidad*, aunque se añadieron algunos otros. Resultó una Metodología moderna y muy bien fundada. En cuanto a técnicas, básicamente usaba las mismas que los demás, ERD, DFD, Diseño Estructurado... A la versión 1 siguió la versión 2... y, por fin, la actual versión 3, desarrollada en la segunda mitad de los noventa, que incorpora ya las últimas novedades (Orientación a Objetos, UML, etc).

Pero Métrica, en ninguna de sus versiones, fue jamás obligatoria en su uso para la presentación de proyectos a la Administración, salvo en algún que otro proyecto aislado, por lo que no tuvo un gran impacto en las empresas españolas. Y, que yo sepa, nadie la usa en nuestros días. Tanto esfuerzo, para nada. Y es una buena metodología, probablemente

las más completa y útil de todas las que pululan (bueno, *pululaban*) por ahí. Pero ya sabemos que los españoles preferimos inventar la rueda cada vez, así que... así son las cosas...

...Porque **el método más usado en los ochenta en las instalaciones españolas fue Method/1**, de Arthur Andersen (aún no se había fundado Andersen Consulting, ni mucho menos Accenture, y, por supuesto, aún no se había destapado el pastel de Enron, así que la compañía todavía existía). Y esto era así por la enorme, apabullante en algunos casos, y hablo por experiencia, presencia de nuestros queridos “*Arturos*” en muchas de las grandes compañías españolas de la época. Y los “*Arturos*” venían con el Method/1 *de serie*, bien aprendido de sus cursos de Chicago, o quizás no, pero en cualquier caso, lo parecía, y convencían a sus clientes (a los Directores, desde luego) de lo increíblemente apropiado de su uso para gestionar costes, conseguir calidad, y bla, bla, bla.

Así que muchísimos profesionales de la época tuvimos que lidiar lo mejor que supimos con los tomos y tomos de fases, actividades, tablas, memorandos, tareas y relaciones interminables del Method/1.

No estoy yo nada seguro de que la aplicación del susodicho método trajera grandes beneficios, excepto para la propia Arthur Andersen, *of course*. Pero nada seguro...

Es más, estoy convencido de que la cantidad ingente de interminables informes, documentos y papeles que había que rellenar no sirvieron para otra cosa que distraer al personal de hacer aquello por lo que se supone que le pagan: desarrollar Sistemas de Información. Una cosa es documentar... y otra, *aquello*. Inhumano. Era inhumano, de veras.

Pero eso sí, si gracias a tu esfuerzo estabas al día con todos tus papeles en regla, cuando el proyecto *se iba* en tiempo, coste y esfuerzo, que siempre se iba... *siempre le podías echar la culpa a otro*, así que fue un sistema perfecto para cubrir las espaldas de los Jefes de Proyecto. Pero desde el punto de vista de las empresas no creo yo que fuera ninguna buena inversión, ni a la corta ni a la larga. En fin.

Ya os habéis podido dar cuenta, supongo, de que adoptar una Metodología sin una herramienta informática que la soporte está abocada al fracaso. Esas herramientas se denominaron *herramientas CASE*, y aparecieron cientos de ellas en los últimos años de los ochenta y los noventa del siglo pasado. Y aseguro que con lo de *cientos* no exagero...

## 14 - Herramientas CASE hasta en la sopa

En el capítulo anterior dimos una ojeada a cómo se fueron definiendo las técnicas de modelización que permitieron formalizar la resolución del Diseño Funcional y Técnico de las Aplicaciones y cómo, a partir de mediados de los ochenta, comenzó su difusión y adopción generalizada. Vimos también que a principios de los ochenta había ya algunas Metodologías de Desarrollo por ahí danzando y cómo adoptaron estas técnicas de modelización rápidamente para formalizar las Fases de Análisis y Diseño. Y también cómo algunos productos informáticos servían para ayudar al desarrollador (hablando con propiedad, al programador) a realizar su trabajo más rápidamente.

Pero estas técnicas de modelización, que estaban muy bien, seguían teniendo una pega: la documentación se generaba con lápiz y papel, lo que originaba un par de problemas: el primero, que realizar cambios a un dibujo hecho a boli mientras se mantienen las aplicaciones no era muy práctico, precisamente (o sea, que casi nadie lo hacía, vaya), y segundo, que... **¡el papel lo aguanta todo!** Allí puedes pintar una relación de  $1:n$  entre un Proceso y una Relación  $n:m...$  y *no protesta lo más mínimo*.

Por lo tanto, sin el concurso de una herramienta informática que se asegure de que el modelo que se diseña es factible y de acuerdo a la técnica, no es posible afirmar que el modelo es, al menos formalmente, correcto. Y la única manera conocida de modificar una y otra vez un cierto modelo, tanto durante su concepción como luego, durante la vida de la Aplicación, y que en todo caso lo que está allí representado sea siempre correcto, al menos desde el punto de vista de la técnica, es guardarlo en formato electrónico y gestionarlo exclusivamente mediante una herramienta informática, un software.

En una palabra, si se quería que todo esto fuera de verdad aplicable, se necesitaban **herramientas CASE** (iniciales de “*Computer Aided Software Engineering*”), y de ellas trata este capítulo.

Supongo que ahora lo normal sería que empiece a contar qué herramientas CASE había, cómo funcionaban, etc... Pues sí, pero será un poco más tarde.

Porque lo primero que quiero reflexionar es sobre **el significado profundo que la implantación de este tipo de Herramientas** junto con las inseparables Metodologías de Desarrollo **tienen en el trabajo del Departamento de Desarrollo**, bueno, en realidad, de todo Proceso de Datos. Voy a hablar de la inescrutable filosofía que está detrás de todas estas técnicas y herramientas de la que jamás he oído yo hablar a nadie, salvo a mí mismo, claro.

Voy a desvelar un secreto...

*¿Qué es, en realidad, una Metodología?*

Una serie ordenada de tareas y procedimientos, junto con las técnicas asociadas, que permiten a los profesionales del Desarrollo de Aplicaciones cumplir con su trabajo, es decir, escribir software y luego implantarlo y mantenerlo funcionando con la máxima calidad y eficiencia, en el menor tiempo factible, y siempre con el menor coste posible. ¿Sí?

Y... *¿para qué sirve una Herramienta CASE?* Para ayudar a realizar estas tareas y procedimientos con un soporte informático que nos ayude, por un lado, a seguir rigurosamente el método seleccionado y, por otro, a asegurar en todo caso y tiempo la corrección formal de nuestro trabajo, la salvaguarda de la información, su mantenibilidad a

lo largo del tiempo... ¿Correcto? Todo esto está muy bien, pero...

¿Con qué objetivo nos pagan nuestros generosos salarios nuestros patronos, cuál es el beneficio que los accionistas de nuestra empresa esperan obtener a cambio de nuestro trabajo, el que pagan con nuestro sueldo? ¿Cuál es, en realidad, el trabajo de nosotros, los informáticos? ...Porque no es para “*Escribir Programas brillantes*”, no. No es tampoco para “*Diseñar Aplicaciones Chulas*”, ni mucho menos, como ya sabéis, ni tampoco para “*Utilizar las Herramientas más novedosas*”, en absoluto.

A nosotros nos pagan para proporcionar a nuestros Usuarios (el resto de la empresa) una serie ordenada de tareas y procedimientos apoyados por un soporte de software informático que les ayude a, por un lado, seguir los procedimientos de trabajo rigurosamente y, por otro, asegurar la corrección formal de la información crítica de la empresa, su salvaguarda, la capacidad de recuperación ante errores, su mantenibilidad en el tiempo...

Volved a leer algunos de párrafos más arriba, si os place, allí donde describía *grosso modo* para qué servían las Metodologías. ¿Veis, quizás, alguna coincidencia?

**¡Exacto! Con el advenimiento de las Metodologías y de las Herramientas CASE, lo que estaba ocurriendo era, de facto, que estábamos informatizando al Departamento de Informática.** Ni más ni menos. *¡Sólo que no lo sabíamos!* O no queríamos saberlo, más bien.

A mí siempre me ha parecido muy curiosa esta esquizofrenia colectiva de nosotros, los informáticos: nos hemos opuesto sistemáticamente a que alguien nos *mecanice* nuestro trabajo, cuando eso es precisamente lo que hacemos nosotros continuamente, mecanizar a los demás... o quizá es, precisamente, *por eso mismo*. Es decir, los pasos lógicos que seguimos cuando nos acercamos a un Usuario cualquiera, el Departamento de Riesgos, o el de Contabilidad, por ejemplo, para poner en marcha una Aplicación para él, siempre han sido, más o menos:

**a) Estudio del Sistema Actual:** En él nos informamos de cómo trabaja ese Departamento, qué datos maneja y qué procesos realiza, qué interfaces tiene con el resto de Departamentos y, en definitiva, cuáles son sus necesidades.

**b) Diseño del Sistema Propuesto:** En base a la información obtenida anteriormente, realizamos el Modelo Tecnológico, el Modelo de Datos que soportará su negocio, el Modelo de Procesos, identificando oportunidades de mejora, etc. Se seleccionan las herramientas hardware y software que se necesitarán para dar soporte al futuro Sistema, se establecen tareas, prioridades, plazos...

**c) Realización del Sistema:** Una vez aprobado lo que se quiere hacer... pues se hace (incumpliendo los plazos, como es tradición), se implanta (con problemas, como siempre), se forma al usuario en su uso (escasamente, claro), se da soporte (como buenamente se puede) y luego se mantiene por los siglos de los siglos, amén.

**d) Implantarlo:** Ponerlo en funcionamiento, formar a los usuarios en su uso, etc.

Por consiguiente, si lo que deseamos es mecanizar al Departamento de Informática, lo lógico es ni más ni menos que **aplicarnos la misma medicina que nosotros aplicamos a nuestros usuarios: Estudio del Sistema Actual, Diseño del Sistema Propuesto, Realización del Sistema e Implantación.** Fácil, ¿no os parece?

**Pues no.**

No conozco ni un solo caso en España, y supongo que tampoco habrá muchos por

ahí fuera, si es que hay alguno, en que esto se hiciera así, es decir, con amplitud de miras, sabiendo cuál es el objetivo final y tomando las decisiones pertinentes en función de ese objetivo. En una palabra, fallamos lamentablemente en el diagnóstico... y por consiguiente, también fallamos en el tratamiento.

La implantación de Metodologías, Herramientas y Técnicas en Desarrollo, y en todo Proceso de Datos, se fue haciendo “a trocitos”, de forma parcial y en muchos casos forzado por la presión mediática y de las consultoras: llegó un momento, a fines de los ochenta, en que “*si no ponías un CASE en tu vida, eras un neanderthal*”, así que se adquirieron herramientas, se adoptaron métodos, se enseñaron técnicas... pero sin un Programa Director que nos indicara de dónde veníamos, a dónde íbamos, ni, desde luego, cuáles eran los objetivos finales a obtener, cuál era la *luz al final del túnel*.

No hubo, en definitiva, metodología alguna que usáramos para la selección de nuestra propia metodología, ni de nuestras magníficas Herramientas CASE... y el resultado es que, **diez, quince o veinte años después de aquellos momentos cruciales en nuestra profesión, no queda prácticamente nada**. Apenas hay alguna gran empresa española que siga hoy en día utilizando alguna de las Metodologías, Técnicas y Herramientas entonces seleccionadas, tras el arduo trabajo de prospección del mercado, prueba, y, desde luego, religioso (¡y abundante!) pago a los proveedores que entonces se hizo.

Siempre hubo una enorme reticencia, *resistencia pura y dura*, diría yo, al cambio por nuestra parte. Nosotros, los adalides del cambio siempre que sea aplicado *a los demás* nos negábamos sistemáticamente (y nos seguimos negando, por lo que sé) a cambiar nuestros usos y costumbres, a aceptar que alguien, aunque sea uno de nosotros mismos, venga a decirnos cómo hacer mejor nuestro trabajo. Porque... *nosotros somos especiales, raritos, únicos y más listos que el hambre, nuestros Sistemas son todos sofisticadísimos, igual que nosotros mismos lo somos, nuestras corbatas son las más bonitas y bla, bla, bla...*

**¡Venga ya!**

La verdad pura y dura es que **hemos fracasado rotundamente en mecanizar** al, supuestamente, Departamento de la Empresa más fácil de mecanizar: **Nosotros mismos**. La *versión 1.0* nos ha salido rana.

Nunca funcionó de verdad, ni siquiera cuando estaba recién implantada. Ayudó poco y entorpeció mucho, o la entorpecimos nosotros, que también. Y feneció de inanición.

Ahora estamos en plena *versión 2.0*: cambio de paradigma (Orientación a Objetos), cambio de Herramientas, de Tecnología... ¿Seremos esta vez capaces de dotarnos a nosotros mismos de un Sistema Informático que cumpla con las expectativas de cualquier Sistema Informático de los que amorosamente entregamos a nuestros Usuarios? No soy yo nada optimista al respecto, pero la respuesta está en vuestras manos, jóvenes amigos. Y *conmigo no contéis esta vez*, que yo ya fracasé cuando fue mi turno.

En fin... después de esta sesuda y sí, un poco amarga reflexión, voy a contar de una vez lo que se supone que debo contar en este capítulo: cómo fue **el baile de las herramientas CASE** a finales de los ochenta y principios de los noventa del Siglo XX.

Ya dije en el capítulo anterior que había ya alguna herramienta a mediados de los ochenta (PACBASE, de CGI y Maestro, de Softlab, eran las más importantes y más implantadas), que servían para ayudar fundamentalmente al programador a realizar su tarea más rápidamente. Y digo *al programador* porque era el único rol de Desarrollo que usaba herramientas informáticas para hacer su trabajo, aunque sólo fuera el editor de programas y

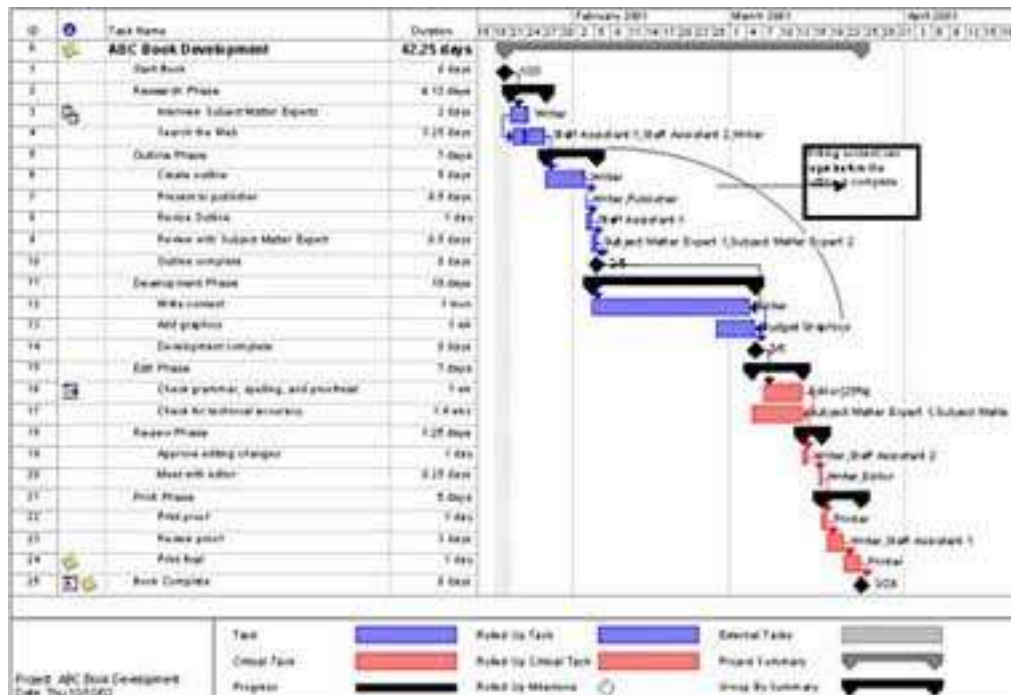
el compilador, dado que el resto de papeles desarrollaban su trabajo mayormente con lápiz y papel.

La propaganda que usaba aquella época Maestro (vendido en España por Philips, antes de que se fusionara con Digital, y luego ésta con Compaq y ésta luego con HP, que ya se había fusionado con Tandem...) lo posicionaba como una *Toolbox*: una Caja de Herramientas diversas que el programador usaba a su conveniencia para escribir o generar código (pero *poco*, ¿eh?), gestionar listados, escribir documentación, etc. Maestro funcionaba en un hardware específico: un miniordenador Philips P7000.

En cuanto a PACBASE, de la francesa CGI, su objetivo era más bien ayudar a seguir implacablemente los pasos de la metodología seleccionada (Merise, vaya, que venía *de serie* con PACBASE, aunque podía usarse o no), y ayudar a escribir programas y generar código, por entonces, poco también. PACBASE era un software que funcionaba en la propia máquina para la que generaba su documentación y su código: un mainframe.

También existían algunas herramientas de Control de Proyectos, como el *MS Project* de ahora, vaya. Artemis era quizá la más conocida. No os extrañará si digo que funcionaba exclusivamente en el mainframe, ¿no? Sin embargo, pensada inicialmente para la Gestión de Proyectos Industriales (construcción, ingeniería, etc.), tenía serios inconvenientes para su uso en proyectos informáticos, pues la asignación rígida de las precedencias entre actividades imposibilitaba controlar prácticas muy comunes en nuestro trabajo como, por ejemplo, que un recurso, analista o programador, comience una cierta actividad sin haber terminado la precedente... esto, por ejemplo, Artemis no lo entendía, así que entre eso y el precio, no debió vender muchas licencias en España para controlar proyectos de desarrollo, por lo que yo sé. Pero para otro tipo de proyectos sí que vendió algunas licencias, que yo lo he visto.

En cualquier caso, según mi experiencia personal, los programas de Control de Proyectos se usan exhaustivamente en la fase de Planificación del Proyecto: se emiten gráficos y diagramas Gantt preciosos como el de la ilustración siguiente, que se reparten, se discuten, se analizan sesudamente, se consensúan, se enmarcan... y, una vez comenzado el proyecto, *jamás se actualizan*.



Si bien todas estas Herramientas fueron las precursoras de las auténticas herramientas CASE (casi, casi, *las inventoras*), no iban a ir por ahí los tiros, por motivos obvios: el advenimiento de las nuevas técnicas gráficas de diseño, por un lado, y la generalización del uso de PC's, por el otro, estaban cambiando rápidamente el foco.

La aparición o, mejor dicho, el éxito de Windows 3.0, el primer Windows que de verdad funcionaba, aunque sobre MS/DOS, eso sí, dotó de una plataforma gráfica *usable* a los desarrolladores de Herramientas Gráficas de todo tipo y también a los desarrolladores de las propias herramientas CASE, que siguieron los pasos del CAD, de *Computer Aided Design*, para realizar programas informáticos gráficos que implementaran las técnicas de Análisis y Diseño.

Sí, había también UNIX para PC en la época, y algunos de estos productos tenían también su versión para entornos UNIX. Ignoro si vendieron mucho o no; no conozco a ninguna empresa que comprara estos productos para UNIX, pero alguna habría, seguro.

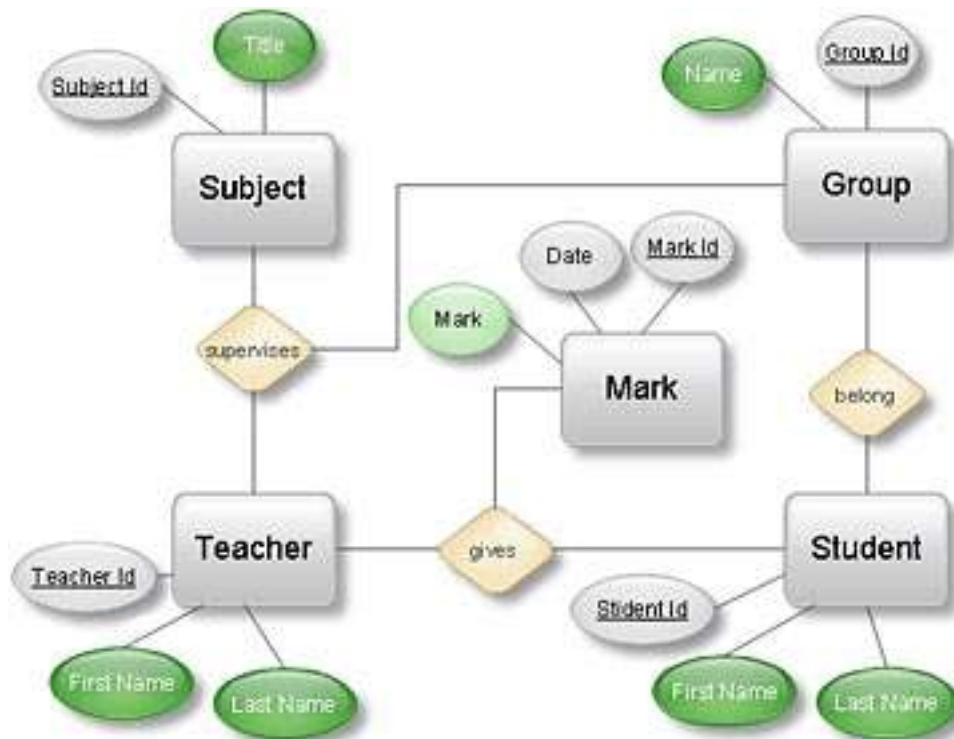
**Una advertencia:** las compañías de software que crearon la mayoría de productos que mencionaré tuvieron una existencia *movidita*. Se compraron unas a otras, luego se vendieron, desaparecieron, aparecieron de nuevo... Es largo, aburrido e irrelevante detallar su historia pormenorizada: los noventa fueron años realmente *interesantes* para los que los vivimos... Así que es posible, incluso probable, que cometa algún error al hablar de estas compañías en lo que sigue. Pido perdón por anticipado, pero es que mi memoria no da para más.

Bien. ¿Cuál de todas fue la primera de ellas?

No lo sé. No me acuerdo, así de simple. Y no es fácil encontrar esa información en la red. Pero en España, de los primeros que empezaron a vender Herramientas CASE gráficas fueron los consultores de Coopers & Lybrand (ahora parte de IBM), que representaban a **Excelerator**, de Intersolv. Intersolv ya no existe, claro, ni su heredera actual vende ni recuerda a Excelerator. Era un software que funcionaba en PC, creo recordar que sólo con MS/DOS y después de 1990, con Windows 3.0 y sucesores.

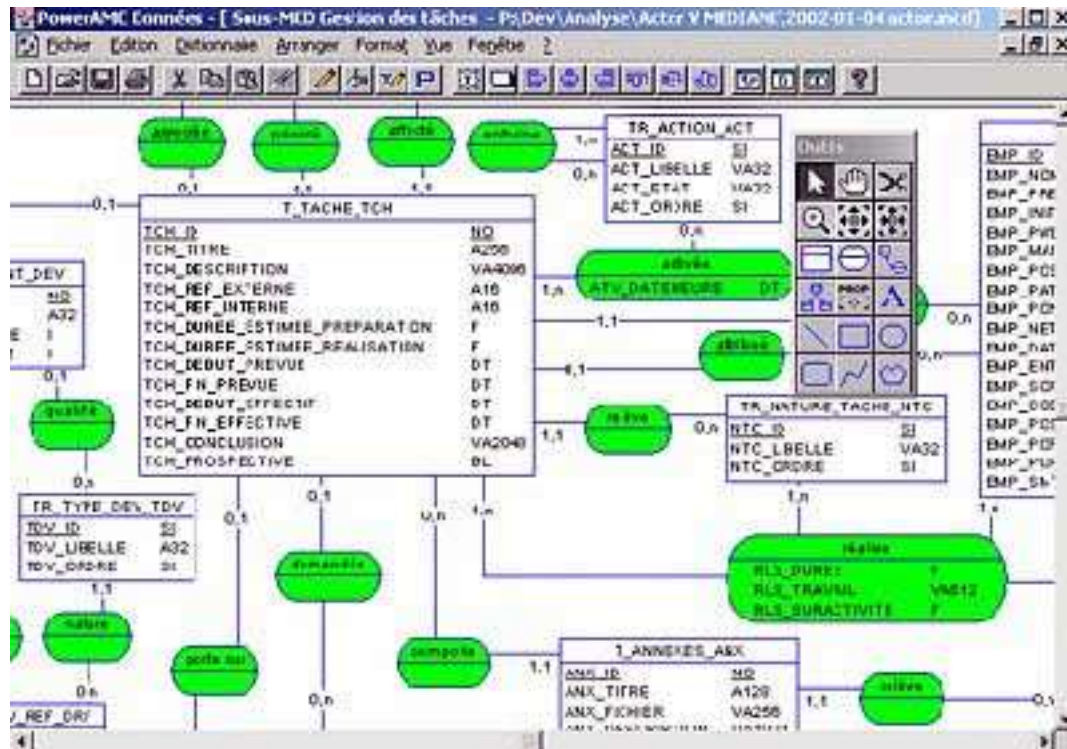


Básicamente tenía adaptadas las dos técnicas de modelización: de Datos (ERD's, de Chen, similares al de la ilustración) y de Procesos (DFD's, de Yourdon), y tenía también un editor que permitía crear un mínimo método alrededor.



Excelerator pintaba *monos* muy bonitos y aparentes, era relativamente sencillo, se colgaba de tanto en cuando y era bastante caro, pues se vendía por usuario y su coste era de varios cientos de miles de pesetas de la época por licencia, quizá llegara incluso hasta el millón de pesetas, es decir, tres mil, cuatro mil, quizá seis mil euros al cambio.

También existía **ADW**, de Sterling Software (tampoco existe ya, ahora es parte de Computer Associates, junto con algunos centenares de otras compañías), y **PowerBuilder**, de Powersoft, adquirida años después por Sybase, que ignoro si sigue en solitario o ha sido adquirida por alguien. Te podían gustar más o menos, pero básicamente hacían cosas parecidas y de forma parecida a lo que hacía Excelerator, todos ellos se colgaban y costaban más o menos lo mismo: **una fortuna**.



En la ilustración tenemos a PowerBuilder diseñando una aplicación con la Metodología Merise.

Otra herramienta CASE más, mucho más modesta y baratita, que apareció algún tiempo después era **Visible Analyst Workbench**, que hacía más o menos lo mismo, pero se integraba peor aún; a cambio era muy baratita comparada con las anteriores, quizá veinte o treinta mil pesetillas de nada (120 a 180 euros) por licencia. Curiosamente, Visible sigue existiendo hoy en día, quizá precisamente porque al ser sencilla y barata, fue capaz de sobrevivir mejor los avatares del fin de siglo.

Y la última que citaré, **Erwin**, de Logic Works (comprada a fines de los 90 por Computer Associates, como tantas otras compañías), es un excelente software especializado en el Diseño de Modelos de Datos para la tecnología relacional que hoy en día sigue funcionando bien.

Había muchas más herramientas CASE. Los últimos ochenta y primeros noventa vieron la aparición de decenas y decenas de herramientas de este estilo, de todo pelaje y condición: no había mes en que no asistiéramos a una presentación o recibiéramos una carta anunciando el lanzamiento de al menos un par de ellas.

Y sin embargo, todas las herramientas CASE de la época basadas en PC, sin excepción, tenían un serio problema: **¿dónde almacenar los datos?** El Diseñador realizaba su Modelo, lo almacenaba en su PC, luego lo copiaba a un servidor usando las poco evolucionadas redes de la época... pero no había forma de coordinar dos modelos diferentes de dos diseñadores distintos, ni tampoco de asegurar cuál era la última versión, ni de garantizar un backup adecuado... En una palabra, **su problema era la falta de integración**, no sólo entre diferentes herramientas, sino incluso entre diferentes licencias de la misma. Claro que tampoco esto es tan raro, porque ahora sigue pasando lo mismo...

Ahora bien, sí que había alguna otra herramienta que era *integrada* desde el

principio, aunque salieron al mercado algún tiempo después de estas pioneras herramientas CASE gráficas.

Entre ellas, a primeros de los noventa, encontramos Foundation, de Andersen Consulting. Básicamente, **Foundation** era un software que funcionaba mitad en PC mitad en el mainframe (no me pidáis más detalles, no los recuerdo) y que era la suma de los diferentes Métodos y procedimientos de la metodología de Andersen: Method/1, Design/1, Install/1 y alguna otra más acabada en el inevitable “/1”.

También **PACBASE** se apuntó a la moda *Cliente/Servidor*, ¡cómo no!, y sacó su versión integrada, donde el front-end se ejecutaba en un PC, mientras que el back-end (el almacenamiento, básicamente, y control de la información), estaba en un servidor, un mainframe, y años después cualquier cosa, incluyendo un Sistema UNIX. Los clientes que tenían PACBASE actualizaron su versión a la nueva arquitectura Cliente/Servidor, allí diseñaron sus sistemas en los noventa... y allí los siguen teniendo. Sí, es un poco *ladrillo*, pero es de los pocos productos que han permitido mantener la información viva a lo largo del tiempo.

Y Softlab hizo evolucionar su Maestro a **Maestro II**, con un diseño similar, hardware dedicado más software, pero cambiando por completo la arquitectura física: el servidor pasó a ser una máquina UNIX de cualquier marca, no sólo de Philips, y el front-end pasó a ejecutarse en un PC con MS/DOS y Windows 3.0. Además, proporcionó un sistema de diseño y programación interna bastante novedoso, que permitía a cada cliente implementar su propia Metodología, o adaptar con cierta facilidad las que tenían ellos ya implementadas: Merise y SSADM, por lo que recuerdo (curiosamente, Alemania, la sede de Softlab, no tenía ninguna Metodología “*nacional*”), y posteriormente Métrica, la metodología española creada por el MAP que fue incorporada en Maestro II a mediados de los noventa.

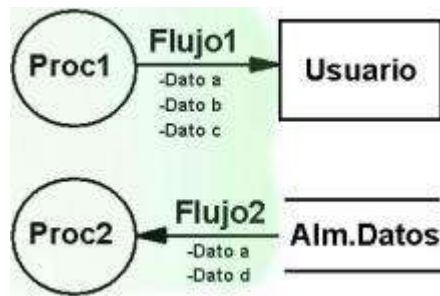
Así era posible realizar el Análisis y Diseño Estructurado con herramientas gráficas y almacenar su resultado en su Base de Datos interna (Orientada a Objetos y funcional... ¡ya en 1992!). Y luego realizar el Diseño Técnico y la Programación (y la generación de los objetos que se decidieran, típicamente las definiciones DB2, CICS, etc) en formato gráfico o de caracteres, según conviniera.

Softlab realizó a principios de los noventa un gigantesco esfuerzo para obtener el **Metamodelo de los Sistemas de Información** (que era una sábana de un metro por medio, por lo menos, pues eran 8 páginas DIN A4, por las dos caras), que es **de lo más interesante para nuestra profesión que he visto nunca**, o mejor, LO MÁS INTERESANTE, sin exagerar, por más que no les ayudó gran cosa como argumento de venta... pocos entendían la importancia de tal documento que, como es costumbre, no hay manera de encontrar hoy por parte alguna...

... ¡Ah, ya! Un... ¿*metamodelo*? Y ¿eso qué es?

Pues es el modelo de los modelos... Ya, ya me explico, ya...

Los Diseñadores realizaban un Modelo (de Datos, de Procesos, etc) que representaban las Bases de Datos, transacciones y programas que iban a dar servicio a los Usuarios.



Por ejemplo, fijémonos en la ilustración anterior. Se trata de un DFD sencillo, muy sencillo, en el que hemos modelizado un Proceso **Proc1** que emite un Flujo de Datos **Flujo1** (con tres datos elementales, los datos a, b y c) a una entidad externa (un **Usuario**) y también otro Proceso **Proc2** que requiere un Flujo de Datos **Flujo2** (con los datos elementales a y d) provenientes de un **Almacén de Datos**. Este modelo es un resultado inmediato y sencillo de aplicar la técnica de Modelado de Flujo de Datos.

Si nos fijamos bien, este modelo consiste ni más ni menos que en representar la realidad, lo que los Procesos deben hacer con los Datos que utilizan, de forma que sea entendible y, muy importante, **almacenable**, si es que este término existe en español. Es decir, podemos capturar los datos sobre el proceso y almacenarlos en un soporte informático de tal modo que podemos reconstruir este Modelo de Procesos a partir de ciertos datos almacenados en un fichero o Base de Datos informático. Eso es, ni más ni menos, lo que hacen todas las herramientas CASE, ¿cierto?

Resumiendo, **el propio modelo diseñado en sí no es, a la postre, nada más que datos**. Ni más ni menos que simples datos guardados en algún soporte.

Pero los datos, del tipo que sean, son susceptibles de ser modelizados, que ya nos lo contó Chen. Para hacerlo disponemos de una técnica poderosa: el Modelo de Entidad-Relación. Luego la consecuencia es que **los propios modelos son susceptibles de ser modelizados**. Eso es lo que se llama un Metamodelo.

Vamos, entonces, armados con la técnica de Chen, a modelizar la parte del modelo que conecta Procesos y Flujos de Datos (la parte sombreada y degradada de la figura de más arriba).

Claramente tenemos aquí tres Entidades: la Entidad “**Proceso**”, la Entidad “**Flujo de Datos**” y la Entidad “**Dato Elemental**”. ¿Cuáles son las Relaciones entre ellas? Un Flujo de Datos puede ser de entrada (entrega datos al Proceso) o de Salida (envía datos obtenidos por el Proceso al exterior). Por tanto, podemos modelizar dos relaciones entre *Proceso* y *Flujo de Datos*: “**LLEGA**” y “**SALE**”.

Ambas relaciones son del tipo *1:n*, es decir, un Flujo de Datos sólo puede llegar a o salir de un Proceso (esto podría quizá ser de otra forma, pero en nuestro ejemplo lo hemos decidido así) mientras que cada Proceso puede (en realidad, *debe*) tener uno o varios Flujos de Datos entrantes o salientes. Y cada Flujo de Datos, por fin, puede a su vez contener muchos Datos elementales, que a su vez pueden estar contenidos en muchos otros Flujos de Datos: se trata, pues, de una Relación del tipo *n:m*.

A continuación tenemos el Metamodelo simple resultado de este proceso de modelización:



Los ingenieros de Softlab hicieron esto mismo con **todos los Objetos posibles con los que trabajamos los informáticos**, tanto los de la parte lógica como los de la física. Por ejemplo, un Programa *usa* una o varias Bases de Datos, una Tabla Relacional *contiene* Atributos (datos elementales), Atributos que *están contenidos por* un Fichero, una Transacción *es atendida por* un Programa, una Entidad *contiene* Atributos... Todos los conceptos posibles, tanto del Análisis como del Diseño o de la Construcción, estaban allí representados.

**Fue una labor ingente**, gigantesca, inolvidable, digna de mejor suerte... o simplemente *de haber sobrevivido*. Yo conservo como oro en paño una copia de este Metamodelo de Softlab, pero es completamente inútil intentar siquiera poner una imagen en estas líneas... no se distinguiría absolutamente nada. Fue un trabajo monumental... e ignorado. ¡A quién se le ocurre hacer esto en *Munich*, en lugar de en *California*!

Naturalmente, las empresas que tenían Maestro (o sea, *Maestro I*) migraron a Maestro II, y Softlab hizo alguna venta más... pero pocas. Se requería un tamaño crítico grande para amortizar los costes; sin embargo, todos aquellos que optaron por Maestro II, o casi todos, siguen hoy en día explotando de una u otra forma la información allí contenida. Fue, a la larga, una buena inversión, largamente amortizada.

Por fin, la cuarta compañía en discordia con herramientas integradas fue Texas Instruments, que, avalada por el ínclito James Martin, creó e impulsó **IEF** (Information Engineering Facility), una herramienta diseñada alrededor de la metodología IE y que, ¡*sorpresa!*!, actualmente es propiedad también de Computer Associates. Ofrecía un entorno fuertemente integrado alrededor de la Metodología patrocinada por el prestigioso Martin, donde todo funcionaba razonablemente bien, siempre que no tocaras una coma al método... y ése era precisamente su problema, porque casi nadie estaba dispuesto a adoptar la Metodología completa tal cual. También consiguió alguna venta en España, pocas, en realidad.

Lo que pasó fue que **la mayoría de empresas que decidieron entrar en el mundo de las técnicas gráficas de diseño, prefirieron hacerlo con software *standalone*** (Excelerator, ADW, Powerbuilder, etc., las que funcionaban en PC), que era claramente menos útil que cualquier software integrado, pero más sencillo de implantar... y que no implicaba tomar grandes compromisos metodológicos de ningún tipo, más allá de *recomendar el uso* del software para hacer los diseños funcionales o técnicos.

El motivo es que realmente hubo muy pocas empresas españolas que de verdad adoptaran una Metodología, fuera de las existentes fuera creada ex profeso para la empresa en cuestión. Ojo, cuando digo "*de verdad adoptaran*" no quiero decir que "*nominalmente adoptaran*"... De éstas últimas sí que hubo muchas. Pero que realmente siguieran siempre y en toda ocasión el método, rellenando todos los documentos exigidos, haciendo y manteniendo todos los modelos... una o ninguna. Y quizá exagero.

Sí que se usaban los Modelos, sobre todo los de Datos: si la herramienta CASE es

buena, por ejemplo, Erwin, PACBASE o Maestro II, una vez realizado el modelo se obtenía el diseño físico preliminar de las Bases de Datos, es decir, hacer el modelo ahorra trabajo en etapas posteriores y la gente lo usó y lo sigue usando, más o menos. En cuanto a los Diagramas de Flujo de Datos, se solían hacer los de alto nivel, sin descomponerlos mucho... y prácticamente nunca se actualizaban una vez terminados.

En cambio, realizar un *Diagrama Ambidextro de Relaciones Impertinentes entre Usuarios Esquizofrénicos* (un suponer), no aportaba gran cosa al Proyecto en sí, daba trabajo, no generaba nada de nada en fases posteriores y requería de mantenimiento posterior cada vez que un *Usuario Esquizofrénico* se convertía en un *Usuario Paranoico Bipolar*. Este tipo de “*Diagramas Imposibles*” (cada metodología al uso tenía *unos cuantos*) creo yo que no los hizo nunca nadie. Y los pocos inasequibles al desaliento que los hicieron jamás los actualizaron...

En este punto posiblemente echéis de menos a alguien... quizá os estaréis preguntando: **¿Y de IBM, qué? ¿Qué pasaba con IBM?**

IBM era el líder absoluto en ventas, tanto de software como de hardware, siempre había tenido un fino olfato comercial y no iba a dejar este prometedor mercado emergente sin incluir en su oferta... y sin embargo, aún no ha aparecido en esta historia... ¿Estarían esos años, por ventura, *dormidos*?

Pues *no*, pero *sí*... pero *no*. *O sea*.

Siguiendo la experiencia exitosa de la creación y lanzamiento de sus PC's, IBM decidió no entrar de lleno en el mercado de la creación y venta de herramientas CASE, sino más bien **proveer un marco para la integración de todas ellas: la Arquitectura SAA** (por *Systems Application Architecture*), que era en realidad un conjunto de normas y definiciones orientadas a permitir la interoperabilidad de todos los productos existentes, con tal que cumplieran las normas marcadas en la Arquitectura.

La idea era genial, muy al estilo de la IBM de la época: como de lo que adolecen casi todas las herramientas CASE del mercado es de integración, **démosle al mercado integración** a mansalva... y de paso, los que cumplan con SAA y entren en nuestro paraguas serán ofrecidos por nuestra fuerza comercial... ¡y nos forraremos a comisiones!

Sí, la idea era genial. Sólo que, por esta vez, *no funcionó*. No tuvo SAA mucho éxito. Más bien poco, en realidad.

Como producto, IBM lanzó su propio entorno integrado, **AD/Cycle**, obviamente en el marco de SAA, que permitía el acceso, en el lado cliente, de cualquier producto CASE y lo que proporcionaba en realidad era la solución al principal problema que tenían las herramientas basadas en PC: **el almacenamiento de los modelos y los datos**, es decir, la anhelada integración entre herramientas.

Para ello lanzó su en la época muy esperado **Repositorio de Información**, lugar que contendría toda la información sobre la información: los metadatos. Algún otro repositorio había ya funcionando entonces, como el de Platinum, otra compañía más de las que, tras adquirir ella misma compañías a *puñaos*, acabó siendo adquirida por... ¡Computer Associates!, cómo no.

Tanto uno como otro, tras la expectación que habían levantado, resultaron un fiasco comercial. Ni vendieron mucho ni tampoco fueron muy utilizados los que se vendieron. No sé exactamente el motivo, posiblemente el mayor de ellos la falta de entusiasmo de los diferentes fabricantes para adoptar estándares comunes que les facilitarían almacenar su información y resolver parte de sus problemas... a cambio de permitir una absoluta

compatibilidad entre herramientas.

Mejor explicar esto con un ejemplo: un cliente cualquiera tiene un puñado de licencias de Excelerator, por ejemplo, y está almacenando los modelos que crea en el Repositorio de IBM, usando el estándar SAA. Pero como ADW, un decir, también cumple a rajatabla con el estándar SAA, mañana ese cliente de Excelerator se enfada con Intersolv, negocia con Sterling, le compra licencias de ADW a buen precio y *con ellas explota*, tan ricamente, *los modelos creados en Excelerator de forma transparente*. Esto a los clientes nos sonaba fenomenal. Y a los fabricantes, *regular*. Tirando a mal... Fatal, vaya. Lo que busca cualquier fabricante es un *mercado cautivo*, no interoperabilidad de ningún tipo con productos competidores.

Así que nunca hubo una interoperabilidad real entre herramientas CASE, por mucho AD/Cycle y mucho repositorio *atómico* que hubiera. El resultado final de todo esto fue un fracaso comercial... de todos.

Ni IBM vendió mucho ni tampoco los fabricantes de herramientas CASE vendieron mucho, a pesar de que todos, pero todos, todos, firmaron acuerdos con IBM para (*supuestamente*) integrarse con SAA... y con el tiempo virtualmente desaparecieron del paisaje.

En fin. No creo que hoy en día quede alguien manteniendo modelos en Excelerator ni en Powerbuilder, entre otras cosas porque debe hacer diez años o más que no hay nuevas versiones. Y del entonces famoso repositorio de IBM... no tengo ni idea de cuál es su estado actual, aunque, conociendo cómo funcionan las cosas en el mundo del mainframe IBM, supongo que habrá unos pocos por ahí instalados... pero de ahí a que se usen de verdad va mucha diferencia.

Parece que ahora, no hace mucho tiempo, IBM se ha dado cuenta de nuevo de la importancia de gestionar la información sobre la información (o sea, los famosos metadatos) que existe en los diferentes Sistemas de Información de las empresas, y ha decidido, tras quince años de abandono, promocionar de nuevo el sucesor "Siglo XXI" de su proyecto repositorio y lo llama *Information Server*, una parte más dentro de su nueva estrategia *Infosphere*... repositorio que, por cierto, no ha sido desarrollado por IBM, sino por *Ascential*, compañía heredera de Informix y comprada por IBM en 2005.

Me pregunto yo: ¿será sólo una estrategia comercial o de verdad hay algo tangible debajo de todo este esfuerzo de marketing? Yo, sinceramente, no lo sé, pero muy optimista no soy. Porque sí que sé que la IBM de la segunda década de los dos mil no es la de la década de los ochenta. Ni mucho menos. Vista su política de adquisición de empresas con productos de toda laya siempre que tengan una cierta base de clientes instalada, creo que se parece mucho a la Computer Associates de los años ochenta y noventa... y esto no le augura un muy buen porvenir, según mi modesta opinión.

En fin, para acabar de liar la madeja, sobre los años 92 ó 93 comenzó a oírse con cierta fuerza en los foros (fuerza que se ha ido incrementando con el paso de los años, qué os voy a contar) la necesidad de un *cambio de paradigma*: había que abrazar la nueva fe de la **Orientación a Objetos**.

El mensaje es claro, diáfano, meridiano: **Todos habíamos estado equivocados durante muchos años**; había que cambiar por completo la manera de Pensar, de Diseñar, de Desarrollar... había poco menos que *hacerlo todo nuevo*. Excelentes noticias... para los Consultores, Fabricantes de Software y, sobre todo, Empresas de Servicios Profesionales de Informática.



Y... ¿para nosotros, los clientes?

Mmmm, creo que no mucho y que, desde luego, las supuestas ventajas del cambio de paradigma difícilmente compensan el coste del cambio... pero ésa es mi opinión personal. Bueno, y la de muchas empresas, que siguen manteniendo sus sistemas críticos en la tecnología “de toda la vida”... pero esa es otra historia y será contada en otro momento.

Lo que sí que quiero citar aquí son algunas de las afirmaciones que se escuchaban entonces y que, en algún caso, se siguen citando ahora, quince o más años después... y todavía no se han hecho realidad.

**1) Desarrollar aplicaciones será en realidad tomar objetos preconstruidos de aquí y de allá y ensamblarlos sin prácticamente código nuevo.** Por ejemplo, tomas el “*Objeto Cliente*”, el “*Objeto Cuenta Corriente*”, etc., que alguien habrá escrito en algún remoto lugar de la Biosfera y luego, simplemente, integras los Objetos entre sí y ¡hala!, ya tienes tu aplicación de Cuenta Corrientes funcionando en un pis-pas.

**2) Próximamente, a los programadores (y diseñadores, etc.) se les pagará más cuantas menos líneas de código escriban,** porque estarán reutilizando código preescrito. Se premiará reusar objetos y se castigará al mediocre que tenga que escribir muchas líneas de código.

**3) Desarrollar software es como construir coches:** en una planta de montaje llegan los componentes individuales y allí sólo se montan, produciendo siempre vehículos de calidad a toda pastilla.

...Y otras semejantes, que os ahorro.

Sobre la primera afirmación, que es idílica, yo siempre me pregunto... ¿y quién escribe el “*Objeto Cuenta Corriente*”, por ejemplo? Supongamos que estamos en el *Banco de Getafe*, en España... ¿tomamos el Objeto Cuenta Corriente preparado por, digamos, el *Banco de Bostwana*? Igual en Bostwana las Cuentas Corrientes no funcionan como aquí. Bueno, tomemos, pues, el susodicho Objeto creado por el *Banco de Alcorcón*, que, vecino nuestro como es, seguro que cubre de perlas mi getafeña problemática. Perfecto, pero, a todo esto, ¿qué opina el *Banco de Alcorcón* de que su competidor, el *Banco de Getafe*, se aproveche de todo el conocimiento sobre Cuentas que ha plasmado en su *Objeto Cuenta Corriente*? Igual no le parece bien... Y si invento un tipo de Cuenta Corriente que el *Objeto Cuenta Corriente* que he adquirido no contempla... ¿qué hago? ¿Lo modifico y me cargo su mantenibilidad futura? ¿Compro otro nuevo? ¿Lo tiro y lo reprogramo todo en Cobol?

No se trata de un problema exclusivamente técnico, que también, sino, sobre todo, comercial, de confidencialidad sobre los productos y criterios básicos del negocio que toda empresa desea mantener ocultos a la competencia a toda costa.

O sea, que podrá haber Objetos de uso general: un buen Quicksort, un objeto para validar fechas, no sé, cosas de ese estilo, por complicadas que sean. Pero ¿Objetos de Negocio, de *Negocio de Verdad*...? Difícil lo veo, salvo en el caso de que estemos hablando de un Sistema de Información Completo, un ERP, vaya.

De la segunda afirmación casi ni hablo: Vete tú a decir a una Empresa de Servicios que tiene que reutilizar todo lo que pueda... cuando lo más probable es que, si vuelve a construir lo que ya está mil veces construido, pueda vender una vez más las horas de desarrollo utilizadas (mejor: *teóricamente* utilizadas) en ello, mientras que si reutiliza y *lo dice*, no hay horas que vender. Y las Empresas de Servicios están para facturar cuantas más horas, mejor, y ganar dinerito, todo el que puedan, no para construir Sistemas de Información maravillosos para sus clientes a costa de arruinarse.

Y, por fin, sobre lo de la fabricación de coches y la Cadena de Montaje... me



pareció siempre una falacia importante. Mucha gente, comerciales y vendedores, sobre todo, repetían como un mantra esta afirmación (*a ver si vendían algo*).

Pero es completamente falsa.

Porque, queridos lectores, **el que se dedica a “fabricar coches” en la Cadena de Montaje es el Usuario Final de la Aplicación. Nosotros, los informáticos, lo que montamos es la Fábrica de Coches, la propia Cadena de Montaje**. Y claro que se puede aprender de las Cadenas de Montaje que se han hecho antes, se pueden reutilizar muchas cosas... pero si el objetivo es fabricar Mercedes Clase E, no nos sirve la Cadena de Montaje de un Fiat Panda, salvo ciertos componentes aislados y no especializados ¿no os parece? Por no hablar del diseño, la aerodinámica, el equipamiento...

Creo que se ve claro, pero por si hay dudas, pongo un *ejemplo tonto*: Sea la Cadena de Montaje de Fiat Panda, que, haciendo un exceso de imaginación, vamos a comparar con la Aplicación de Nómina.

Todos los Fiat Panda son básicamente iguales, aunque pueden llevar montados un par de motores diferentes o tres, diversas tapicerías y estar pintados en dos docenas de colores distintos. También todos los empleados contenidos en la Aplicación de Nómina son básicamente iguales (*sus datos*, sólo *sus datos*, no me malinterpretéis...), unos tienen una categoría, sueldo y condiciones laborales, y otros, otras, pero los datos de todos ellos están guardados en la misma *Cadena de Montaje*, huy, perdón, Aplicación de Nómina.

La Cadena de Montaje recibe instrucciones para la fabricación de cada coche individual: éste, Verde Botella, con motor de gasolina 1.3 y tapicería *yes-very-well*, el siguiente rojo, Diesel y tapicería de cuero... Instrucciones concretas dentro de una lista de posibilidades reducida: sólo podemos fabricar Fiat Panda en esa Cadena, no Fiat Croma, ni menos aún Nissan Patrol...

La Aplicación de Nómina, por su parte, procesa a los empleados según sus condiciones particulares, si tiene jornada reducida, si tiene complementos de tal y cual cosa, etc., dentro de una serie reducida de posibilidades y les calcula su recibo de nómina... pero no puede calcular la Venta por Región, ni los Ratios por Sucursal, ni realizar transferencias, porque eso lo hacen otras aplicaciones que para eso están. En realidad, ni siquiera podría calcular la Nómina de otra empresa diferente, con convenio laboral, categorías y condiciones de trabajo diferentes, complementos diferentes, etc.

*Fin de la analogía tonta.*

Además, para acabar de complicar el asunto, a principios de los noventa comenzaba a ser evidente, curiosamente, la presencia de una... no sé, una *contestación* al caro *Sistema Metodológico* imperante.

Por una parte, los consultores y gurús nos decían que había que crear o adoptar una Metodología, cumplirla a rajatabla, controlar actividades y tareas (y costes, claro), documentar todo lo habido y por haber...

Y por otra, en algunos casos los mismos consultores y gurús que nos vendían la recojo-metodología, como James Martin, que promocionaba activamente IEF con su metodología IE detrás, pero que simultáneamente era el adalid del *RAD*, nos decían que desarrollar *todos los proyectos* de esta guisa era, a su vez, muy costoso en tiempo y dinero, así que había que inventar algo para desarrollar rápidamente pequeños Sistemas de Información que no requieran tanta formalización... y la consecuencia es que nos vendieron el **RAD** (Rapid Application Development).

Ya unos años antes se pusieron de moda los “Lenguajes de Cuarta Generación” o

4GL, que permitían realizar programas sencillos, listados sobre todo, sin necesidad de tanta línea de código, estructuración y todo el resto de parafernalia que sería necesaria programando en la tecnología normal, en la época, Cobol o PL/1, mayormente. El primero fue MARK/IV, luego vinieron Mapper, Nomad, Focus, Easytrieve, Mantis, Ramis, Clipper... y un larguísimo etcétera.

No voy a hablar mucho aquí de estos 4GL's. Sólo decir que, a pesar de que todos ellos prometían un espectacular aumento de la productividad, del orden de ocho o diez veces o más, en la práctica no era tanto... y en ocasiones no sólo no se ganaba nada, sino que se perdía productividad y siempre, siempre, se perdía rendimiento, pues eran mucho menos eficaces en cuanto a uso de recursos de máquina que un buen programa Cobol. A la larga, el más exitoso de ellos ha sido **Natural**, de Software AG, que funciona perfectamente bien tanto accediendo a la Base de Datos Adabas como con el resto de Bases de Datos Relacionales. Natural ha ayudado mucho a Software AG a resistir con solvencia tanto cambio tecnológico habido desde entonces.

Bueno, pues esto del RAD venía a ser una vuelta de tuerca en la misma dirección que los 4GL: mucho Usuario o Director de Proceso de Datos vio una oportunidad para aligerar costes y ganar tiempos... Se vendieron productos y metodologías RAD, en muchos casos a los mismos clientes que compraron grandes Metodologías Estructuradas unos pocos años o meses antes... A muchos nos gustaban muchísimo estos nuevos métodos RAD, porque no era, ni más ni menos, que oficializar el método "*a la mecagiéndiez*" de toda la vida.

Se compraron productos RAD de esos, se implantaron, se hicieron proyectos... En realidad, se convirtieron en un *coladero* que sirvió para desarrollar proyectos que nunca jamás deberían haber sido escritos en un 4GL auspiciado por un buen RAD, ahorrando quizá algo de tiempo y dinero en su creación, pero que se convirtieron en una pesadilla para mantenerlos en cuanto estuvieron en Producción.

¿De verdad creéis que se ganó mucho? El tiempo que se ganó dejando de documentar, que tampoco se ganó tanto, se perdió luego con los mil y un problemas en la implantación. El mantenimiento era un caos, tanto que en muchos casos compensaba reescribir por completo el código antes que bucear en el proceloso mar del código existente (*¿os suena de algo a vosotros, informáticos de las nuevas hornadas, esta manera de proceder con el mantenimiento de Aplicaciones?*).

Lo que sí es seguro es que James Martin, entre unas cosas y otras, ganó muchísimo dinero, tanto como para poder crear una Fundación con su nombre en la Universidad de Oxford y dotarla con ciento cincuenta millones de dólares de su peculio, gesto que le honra.

Y en cuanto a nosotros, a veces atacábamos los proyectos con nuestra poderosa metodología estructurada, con sus técnicas y métodos, y a veces decidíamos que no merecía la pena meterse en tales berenjenales y aplicábamos el paradigma RAD... o sea, ningún paradigma en absoluto. El criterio para elegir una u otra forma era, mayormente, las prisas, la presión de tiempos y costes, las querencias de cada uno... Criterios siempre muy *profesionales*, como podéis colegir.

En fin, han pasado los años... y de todo esto... *¿qué?*

¿Qué queda hoy en día, **cuánto de ese esfuerzo realizado en los años finales de los ochenta y todos los noventa sirve hoy para algo?**

Yo diría que sí sirvió, y mucho. Nos acostumbró a seguir un método sistemático en el Desarrollo de Aplicaciones (o, al menos, a concienciarse de que los métodos existían,

que algo es algo); nos introdujo en nuevas formas de hacer las cosas, diseñando antes que programando, planificando antes de lanzarse al río de cabeza... Nos acostumbró a utilizar y *perder el miedo* a las nuevas herramientas gráficas, nos enseñó a nosotros, los creadores de Sistemas de Información, que había en el mundo algo más que pantallas de caracteres 80x24 para plasmar las aplicaciones, que había otra forma de hacer las cosas, que también podía haber colorines y ratones en las pantallas de nuestras aplicaciones. Nos fue convenciendo poco a poco, subliminalmente casi, de que otra forma de trabajar era posible.

Las herramientas CASE, durante los noventa, nos fueron preparando mentalmente para aceptar la revolución que se acercaba... pero esa es otra historia y será contada en otro momento. Obviamente, las herramientas que se usan hoy en día son, en su mayoría, Orientadas a Objetos. Pero aún hay en Producción miles y miles de Aplicaciones desarrolladas con métodos estructurados que hay que mantener funcionando, modificar, incorporar las nuevas necesidades... pero yo creo que rara vez se usa para esto la herramienta con que se diseñó originalmente el Sistema. Se tocan las Bases de Datos, los Programas, los Ficheros, se documenta malamente, o nada en absoluto, y punto.

O sea, que podríamos tirar tranquilamente todos los modelos amorosamente guardados durante años,... ya no reflejan la realidad, lo que de verdad está instalado en Producción, y como decía mi primer jefe en mi primer trabajo en mi primer Banco: *“para tener la documentación desactualizada, más vale no tener ninguna documentación”*: te ahorras el tiempo de buscarla, mirarla y comprobar que no refleja la realidad y te ahorras, de paso, el enfado que pillas cuando te das cuenta de que estás perdiendo el tiempo. Y, total, *como de todos modos no piensas actualizarla...*

Herramientas CASE hay ahora mismo a cientos. Ya no conozco casi ninguna, pues hace bastantes años que no sigo estos apasionantes temas, y las que conocí bien... ya no existen, o a lo mejor sí que existen, pero como si no.

¿Se utilizan? En mi opinión, *sí pero no*. Se usan, al menos relativamente, para crear los modelos y las especificaciones iniciales de la aplicación, para hacer las generaciones iniciales de las Bases de Datos, etc. Y, una vez puesta en Producción la Aplicación, ya se usan poco. O nada. O quizá sí que se mantenga en ciertas aplicaciones e instalaciones... Seguro que, como siempre, *de todo hay en la viña del Señor*.

En los capítulos que siguen cambiaremos de década; dejaremos la emocionante década de los ochenta para entrar de lleno en la tormentosa y no menos emocionante de los noventa.

El próximo capítulo estará dedicado a las convulsiones tecnológicas, pero sobre todo comerciales que acontecieron durante esos años, donde vivimos la rebelión de los fabricantes pequeños contra el amo y señor, la antigua *madrastra de los enanitos*, que terminaron finalmente por convertirla a ella misma en un *enanito* más...

## 15 - Los años noventa (I): La Guerra de los PC's

El necesariamente incompleto periplo por tres de las tecnologías clave para la Informática de hoy en día que tuvieron su amanecer durante la década de los ochenta terminó en el capítulo anterior: se trata del advenimiento de los PC's y su rápida generalización, de la aparición de las Bases de Datos relacionales y su no menos rápida generalización y de la llegada, muchas veces asfixiante, de las Metodologías de Desarrollo y de sus acólitos, los innumerables productos y herramientas CASE que nos animaron la existencia en los años ochenta y primeros noventa.

Dejamos atrás los ochenta y a partir de ahora nos moveremos por los *singulares años noventa*, que, como indica el título del capítulo, fueron ciertamente tormentosos en el mundo de la tecnología informática, sobre todo en lo que respecta a los movimientos alrededor de los PC's y sus Sistemas Operativos, de los que hablaré en este capítulo, mientras que en el próximo me centraré en el resto de convulsiones tecnológicas de esos años: los cambios habidos en el mundo de los “minis” y de su software, que propiciaron enormes cambios en la forma misma de entender el papel de cada elemento tecnológico en nuestro mundo. Aunque, en realidad, **todo esto ocurría simultáneamente**, y otras muchas cosas más, haciendo que la bendita profesión informática resultara durante todos esos años algo *realmente interesante*...

No esperéis, como vengo repitiendo, ninguna crónica oficial: contaré lo que a mí me parece más relevante e interesante, siempre desde el punto de vista de uno que pasaba por allí... Me dejaré muchas cosas en el tintero, igual alguno de vosotros, lectores, no estáis de acuerdo con algo, o incluso de mucho de lo que escriba... ¡qué le voy a hacer!

A mediados y finales de los ochenta IBM dominaba de largo el mercado de PC's, en base a su pequeño *Frankenstein*: el IBM PC, al que había sustituido el PC XT (por eXtended Technology), y hacia 1985, el PC AT (por Advanced Technology).

Su éxito, como vimos entonces, se debió a un cúmulo de circunstancias, entre las cuales destacan dos: una, la enorme capacidad de marketing del IBM de la época, y dos, el hecho de que las especificaciones del IBM PC fueran abiertas, es decir, que estuvieran publicadas y fueran accesibles para quien quisiera.

En efecto, cualquiera podía construir placas o dispositivos para conectarlos a un IBM PC, incluso cualquiera podía comprar procesadores a Intel u otros fabricantes que los constrúan bajo licencia y construir su propio PC clónico y competir con la propia IBM... y justo esto mismo es lo que hacían cada vez más fabricantes.

Porque IBM, tras el fiasco que supuso el lanzamiento del PC portátil IBM 5100, y en su afán de sacar a la venta un producto que pudiera competir en el emergente mercado con Apple y el resto de competidores, cometió algunos fallos... aunque no fueron evidentes hasta algunos años más tarde. De hecho, en los *años de vino y rosas* donde se vendían carísimos PC's de IBM como churros nadie pensaba en estos fallos, es más, absolutamente nadie *sabía* siquiera que pudieran ser *fallos*.

El primero de ellos fue **infradimensionar gravemente el mercado al que se enfrentaba**: las previsiones de venta de IBM para su nuevo IBM PC fueron inicialmente de 250.000 unidades... en cinco años. Después de su anuncio, y antes incluso de que se

comenzaran a servir PC's a los clientes, IBM había ya sobrepasado con creces esa cifra de pedidos. Y esta infravaloración del mercado pesó como una losa en muchas de las decisiones de diseño.

Otras veces he dicho, y lo repito ahora, que los de IBM no tenían un pelo de tontos. Si hubieran tan sólo entrevisto el tamaño real del mercado a que se enfrentaban jamás hubieran tomado las decisiones que tomaron. Pero, claro, ellos sabían muchísimo de tecnología y de soluciones empresariales, puesto que prácticamente habían inventado muchas de ellas... pero no sabían prácticamente nada sobre artículos de consumo. En realidad, **a los responsables de IBM les pilló completamente desprevenidos el éxito de su invento.**

Así, como resultado de ese error en el dimensionamiento del mercado, **ciertas decisiones de diseño que se tomaron no resultaron adecuadas a la larga** (*adecuadas* hablando desde el punto de vista de IBM, desde luego). Quizá sí hubieran sido correctas si hubiera sido acertado ese volumen previsto de ventas, pero no con volúmenes decenas de veces superiores.

Una de estas decisiones *incorrectas* fue, paradójicamente, parte de la clave de su éxito: **las especificaciones abiertas**. IBM sólo mantuvo como propiedad industrial las especificaciones de la BIOS, pensando que todo aquel que deseara fabricar un PC competidor debería pagarles royalties por usar esa BIOS. Pero luego diversos fabricantes hicieron ingeniería inversa y construyeron su propia BIOS compatible, con las mismas especificaciones y funciones o muy parecidas, pero con distinto código. Las leyes del copyright no protegen la copia de las especificaciones de uso, que, además, eran públicas, ni tampoco las imitaciones, sino la copia literal del código, y esto no se había producido, así que IBM no pudo ganar ninguna demanda al respecto.

La segunda *equivocación* de IBM, y crucial, fue **no escribir su propio Sistema Operativo**. Seguramente debido a la premura de tiempo por sacar un producto al mercado, decidieron usar (y, si fuera posible, comprar) un Sistema Operativo ya creado y no escribir un Sistema Nuevo, que es lo que siempre había hecho. **Ésta fue, en realidad, la decisión clave...** pero ¿quién podía saberlo en 1980?

La decisión obvia hubiera sido utilizar **CP/M**, de Digital Research, pero ambas compañías no llegaron a ningún acuerdo. IBM ofrecía sus condiciones leoninas standard, *as usual* (parece que querían pagar una cierta cantidad, no demasiado, por adquirir la propiedad del producto y luego pasar una pequeña cifra a Digital Research por cada licencia vendida), y Gary Kildall, el fundador y presidente de Digital Research, que gozaba por entonces de una desahogada posición y un confortable margen, pues era líder absoluto en Sistemas para ordenadores domésticos, les dijo a los de IBM que... *que se alegraba de verles buenos*.

Y no hubo acuerdo, obviamente, u hoy en día la historia sería diferente... no sé si mejor o peor, pero diferente.

Acudió entonces IBM a Microsoft, donde Bill Gates, uno de los mejores comerciales que ha visto el siglo, les aseguró que *Microsoft sí podía* proporcionar a IBM un Sistema Operativo para su nuevo PC y a tiempo para el lanzamiento. Aquí las versiones divergen: que si compró un clon de CP/M; de nombre QDOS, por cuatro perras (sin decir para qué, claro) y luego se lo encasquetó a IBM, que si lo fabricó de la nada en cuatro días, que si una mezcla de ambas...

En España no nos enteramos de nada de nada hasta mucho después, así que sólo puedo dar mi opinión de oídas y de *leídas*, pero la que me parece más sensata es la versión

de que Microsoft compró QDOS, adaptó quizá cuatro tonterías, y lo rebautizó como MS/DOS para que luego IBM lo re-rebautizara como PC-DOS. La razón es que en aquel tiempo Microsoft estaba empeñada en migrar UNIX para su funcionamiento en ordenadores de 16 bits: el Xenix, proyecto que resultó un completo fracaso y fue abandonado, mejor dicho, cedido poco tiempo después a Santa Cruz Operations, que fue quien adoptó, terminó y lanzó este Sistema años después, así que no creo yo que en 1980/81 estuviera el horno de Microsoft para otros bollos que no fueran el dichoso Xenix.

**Da igual, en realidad.** El hecho es que, en 1981, el Sistema Operativo del IBM PC primigenio era el PC/DOS versión 1.0, e IBM no impuso a Microsoft las *leoninas condiciones* que fueron en su momento rechazadas por Digital Research, permitiéndole mantener en este caso la propiedad del Sistema, aunque compartida de no sé qué modo, debido a alguna razón para mí desconocida, aunque sí puedo aventurar mi favorita: la habilidad negociadora de Mr. Gates jugando con la premura de tiempo de IBM. A principios de 1981, Microsoft se había convertido ya en la única alternativa válida de IBM para ser capaces de lanzar el producto a tiempo, y seguro que Gates sacó partido de ello.

Microsoft durante los primeros tiempos fue, cómo no, un perrito faldero de IBM... hasta que sacó los dientes, unos años después. Y... ¡*menudos bocados que daba!* Pero no nos adelantemos...

¿Cómo era aquel primer MS/DOS (o, lo que es lo mismo, PC/DOS)? Pues... a duras penas se podría llamar a *esa cosa* un *Sistema Operativo*. Sería más bien un mero programa que funcionaba en la máquina y proporcionaba servicios básicos para que otros programas se ejecutaran allí. Pero tenía serios inconvenientes...

**...Porque ni de lejos cubría, ni el propio MS/DOS ni tampoco la BIOS, toda la funcionalidad del hardware del PC.** Por ello, cualquiera que quisiera sacar todo el partido posible al hardware debía acceder directamente a los dispositivos, dejando a un lado tanto al Sistema Operativo como a la BIOS.

Informáticamente hablando, esto es *una barbaridad sin paliativos*, pero durante muchos años se han desarrollado así las aplicaciones, así que a estas alturas casi nos parece hasta normal. Y no lo es, vaya si no lo es. Por descontado, tampoco lo era en 1980: me sorprende que IBM, autores del mejor Sistema Operativo de la historia, qué digo, *inventores* del concepto de Sistema Operativo moderno, tragara con semejante bodrio. Pero tragó. Y MS/DOS fue el “*Sistema-o-lo-que-sea*” más vendido, con gran diferencia, entre 1981 y 1995.

Como el MS/DOS era como era, estábamos condenados a:

1) Que los usuarios sólo tuvieran acceso a una tarea: era un **Sistema mono** (no que fuera “*mono*” de “*bonito*”, que era bastante feo, sino de “*monotarea*”). El hardware hubiera permitido la multiprogramación, de hecho la permitió pocos años después, pero así eran las cosas entonces.

2) Que **los diferentes programas no fueran bien unos con otros**. Había problemas de incompatibilidad por todos lados, consecuencia de la “*rudeza*” del API (interfaz para el programador) tanto de BIOS como de DOS; diferentes fabricantes podían adoptar, y adoptaban, soluciones diametralmente distintas para resolver el mismo problema, con lo que *la incompatibilidad era la norma*.

3) Que **sólo se pudiera direccionar hasta 640Kb**. Y en “modo real”, puesto que DOS no tenía “modo protegido” y no lo tuvo hasta años después, cuando lo soportó el

hardware. Según parece, Bill Gates decía en 1981, en una de sus famosas frases, que “640Kb deberían ser suficientes para todo el mundo”... pero no dijo *cuándo* dejarían de ser suficientes. Y a mediados de los 80 ya no eran suficientes. Así que, para poder direccionar más memoria, se inventaron más y más artefactos: la memoria extendida, la memoria expandida y qué sé yo qué más.

4) La consecuencia final de todo esto: **MS/DOS era inestable. Muy inestable.** El gesto de pulsar simultáneamente las teclas Alt+Ctrl+Del se convirtió en un gesto universal para indicar “empieza de nuevo, amigo”: entonces, la famosa combinación de teclas directamente rearrancaba el ordenador... o no, en cuyo caso se requería de un buen “*botonazo*”, lo que ocurría con bastante frecuencia, por cierto.

En definitiva, escribir aplicaciones que usaran toda la capacidad de algún hardware concreto, una placa, un dispositivo, cualquier cosa, era *una locura*. De frenopático.

Recuerdo que tuvimos que realizar una aplicación para controlar el software que se enviaba a las oficinas, es decir, las nuevas versiones del programa de las oficinas. Antes se enviaban por valija (o sea, por correo, en cintas magnéticas o en disquetes, más adelante en CD's), pero con la paulatina mejora de las comunicaciones se hizo posible pensar, al menos *pensar* en repartir el software por vía telemática.

Podéis imaginar que en tales fechas, finales de los ochenta, no había software alguno en el mundo mundial para hacer tal cosa, así que nos arremangamos y nos pusimos a escribirlo. Tampoco es tan raro, ¿no?, que, al fin y al cabo, *somos informáticos y nos pagan por escribir software*. Seleccionamos una placa de emulación 3270 (quizá una IRMA, aunque no estoy seguro) que tenía la ventaja de incorporar un API potente para interactuar con ella.

Fichamos también a un par de personas que supieran C, que no nos fue nada fácil encontrarlos, por cierto, para poder programar con el dichoso API. Entre los centenares de profesionales de Informática de la empresa no había un solo programador que supiera lo suficiente de C, ni casi tampoco había fuera. Hecho el diseño, se comenzó la programación. La parte correspondiente al mainframe no tuvo el menor problema y estuvo lista en su momento.

Pero en el lado PC... tuvimos meses de retraso: aquello no había modo de que funcionara, la documentación era incorrecta, lo que se suponía que había que hacer de una forma concreta fallaba, y había que probar, por ensayo y error, hasta encontrar algo que funcionara más o menos... Los programadores se deprimieron y, por fin, cuando parecía que ya funcionaba todo, la nueva versión de la tarjeta traía otra API diferente, así que no valía el desarrollo hecho y había que volver a hacerlo... Un desastre.

Nunca nos funcionó el envío telemático de software hasta que no pasaron unos añitos. Y sobre todo, hasta que la tecnología evolucionó lo suficiente y hubo productos que fueron capaces de hacerlo.

Volvamos a nuestra historia.

Ya hemos visto que el DOS (PC/DOS o MS/DOS, a vuestro gusto), no era ninguna joya. Y los ingenieros de IBM lo sabían perfectamente... ¡quién mejor que ellos para saber lo que debería hacer un Sistema Operativo! Además, habiendo constatado el error de cálculo cometido, IBM comenzó inmediatamente a tomar medidas para repararlo y volver a controlar en la medida de lo posible tan grande, rentable e *inesperado* mercado. Tomaron la iniciativa por dos vías diferentes, aunque complementarias: por un lado **atacaron el**

**hardware**, para volverlo al redil al que estaban acostumbrados: los diseños propietarios, protegidos por patentes de IBM, que les aseguraran pingües beneficios (después hablaré de esto); y, por el otro, **poniendo orden en el desastre del Sistema Operativo**, corrigiendo en lo posible sus fallos...

...Pero manteniendo, eso sí, una máxima de diseño: **había que mantener la compatibilidad hacia atrás**. Es decir, cualquier cosa que funcionara en la versión 1 debería funcionar en la versión 2, la 3... y en todas: era **la marca de la casa**, e IBM no renunció a ello. Visto en retrospectiva, hubiera sido mucho más eficaz y lógico tirar toda aquella basura y hacer algo nuevo como Dios manda, sobre todo a partir de la introducción en 1985 del IBM PC/AT, que ya portaba un flamante procesador Intel 80286 con mucho mejor soporte hardware, un nuevo bus, más interrupciones, un reloj de tiempo real integrado, mejores dispositivos... pero así fueron las cosas.



Además, el nuevo procesador 80286, del que tenéis una foto en la ilustración (se trata de un 80286 fabricado bajo licencia por Harris), aportaba una mejora muy significativa: el soporte del modo de operación “protegido”, además del soporte del modo “real”, que era en realidad el único modo posible en los 8086 y 8088 de anteriores modelos.

Es decir, ahora el Sistema Operativo podría fácilmente asegurar (bueno, vale, *casi* asegurar) mediante el hardware que una tarea no accediera, por error o deliberadamente, ni a las propias instrucciones del Sistema Operativo ni tampoco a otras aplicaciones: se podía limitar la *barra libre* que había habido hasta entonces, donde cualquier programa podía acceder sin cortapisas a cualquier dirección de memoria que le diera la real gana, pues eso exactamente es lo que implica el “modo real”, que ahora queda restringido al Sistema.

Tanto IBM como Microsoft pusieron manos a la obra para mejorar el software; tan pronto como a principios de 1983 Microsoft arrancó el desarrollo de un Sistema Operativo multitarea “*serio*” que, tras varios cambios de su nombre, acabó por llamarse, algunos años después, **OS/2**. Y entre medias Microsoft lanzó Windows en 1985, con la idea de proporcionar un entorno gráfico multitarea a los usuarios de DOS, supongo que mientras cristalizaba el nuevo Sistema.

La versión 1.0 de Windows era un *bodrio infumable*, entre otras cosas por serias limitaciones en la gestión de las ventanas, limitaciones impuestas por reclamaciones de infracción de patentes por parte de Apple. Por ejemplo, las ventanas no podían solaparse unas sobre otras, porque Apple mantenía que tenía patentado ese mecanismo; tampoco tenía Papelera de Reciclaje, ya que Apple aseguraba que tenía patente sobre ese concepto... Sí, *a mí también me hace gracia*... ¿cómo puede *patentarse* que una ventana no pueda tapar a otra? Pero estaba patentado, ya veis, cualquier cosa es posible en EEUU; sin ir más lejos, hay una compañía que tiene patentado el XOR:  $0+0=0$ ;  $0+1=1$ ;  $1+0=1$ ;  $1+1=0$ . ¡No lo uséis, que está *patentado*! Ay, si Boole levantara la cabeza...

Así que Windows 1.0 era una castaña, y la versión 2.0, de 1987 tampoco era mucho mejor...

El caso es que al desarrollo del futuro OS/2, que por entonces tenía otro nombre,



vaya Vd. a saber cuál, porque debió cambiar como seis veces de denominación, se sumó IBM alrededor de 1986, firmando un “*Joint Development Agreement*” con Microsoft que daba a ambas compañías la propiedad conjunta sobre el producto resultante y garantizaba a ambas el acceso al código que la otra realizara.



El nombre final, **OS/2**, se le dio justo antes de su salida al mercado, a principios de 1988, aparentemente para asociarlo a la nueva gama de ordenadores personales de IBM, es decir, la gama IBM **PS/2**, por “Personal System/2”.

Lo de poner el sufijo “/2” a los productos de IBM fue una constante de marketing de fines de los ochenta, empezando por **DB/2**, que era la segunda (y mejor) Base de Datos, **PS/2** sería la segunda (y mejor) generación de ordenadores personales, y **OS/2** era el segundo (y mejor) Sistema Operativo para PC.

Estos dos últimos, *mejores* eran. Pero mucho. Y, a la larga, fracasaron comercialmente. ¿Os acordáis del Video 2000, el Beta y el VHS? En fin.

En mi modesta opinión, creo que la elección del nombre OS/2 no fue nada adecuada a la larga. El famoso “/2” lo asoció a IBM casi indisolublemente, mucho más de lo que en realidad lo estaba. Seguramente esto es lo que buscaban los genios del marketing en 1987, apoyándose en el prestigio de la compañía líder, pero algunos años más tarde esta misma asociación, *OS/2 = IBM*, contribuyó a su fracaso de alguna manera.

OS/2, aunque obviamente basado en el código de MS/DOS, traía espectaculares mejoras: Multitarea preemptiva (toma ya *palabreja*: quiere decir, más o menos, multitarea “de verdad”); Multithreading, Soporte para memoria virtual; Direccionamiento de hasta **16 Mb** de memoria real!, etc. Mejoras todas ellas muy importantes sobre el MS/DOS de la versión que fuera. Había sido escrito prácticamente todo él en Assembly, puesto que debía ser capaz de funcionar en un PC/AT con no más de un Mb de RAM, así que era un Sistema Operativo realmente rápido y eficiente.

Sin embargo, el principal problema con que se encontraron los diseñadores de OS/2 fue, curiosamente, ser capaces de mantener la compatibilidad con el dichoso MS/DOS, sobre todo las aplicaciones que accedían directamente al hardware, que eran muchas. Pero no les quedaba más remedio que soportar esas aplicaciones MS/DOS debido al enorme número de licencias vendidas existentes en el mercado, muchas de ellas vendidas por IBM, a las que había que dar continuidad **sí o sí**: recordad, ésa era la *marca de la casa*... era. Porque ya no.

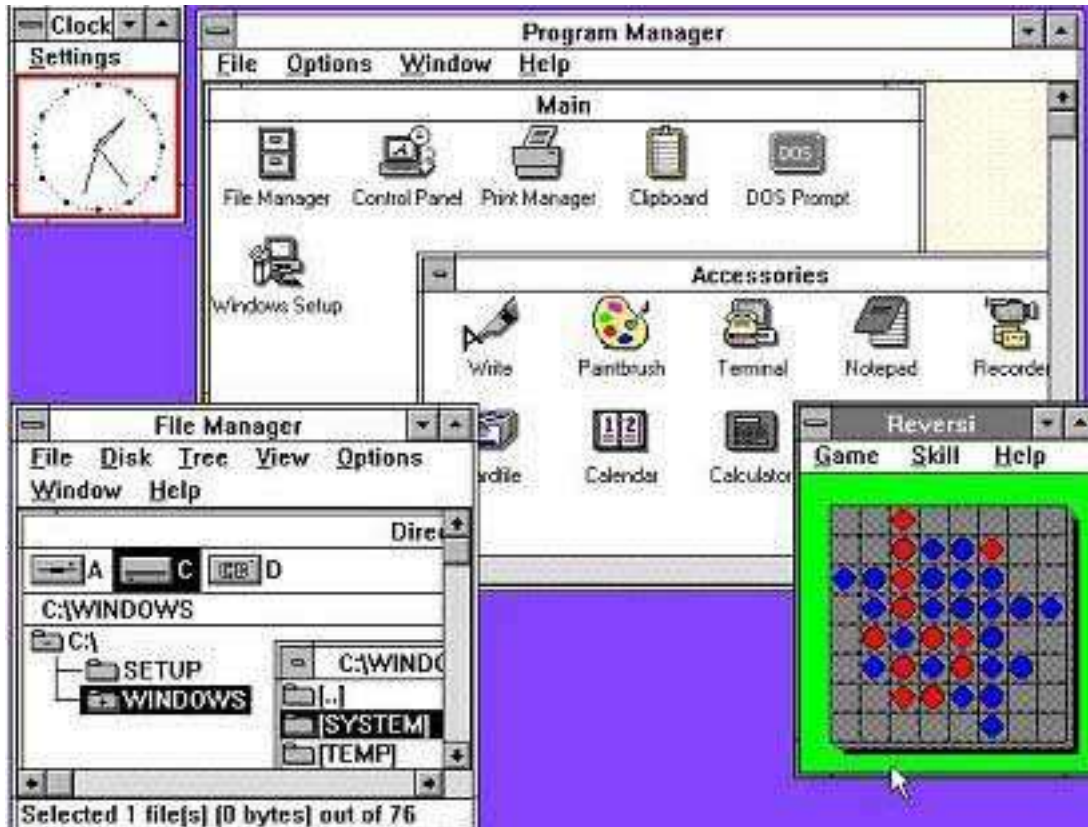
Esta versión 1.0 de OS/2 seguía siendo en modo texto: el modo gráfico que debería proporcionar Presentation Manager, el equivalente *Oesedero* del Windows, no estuvo disponible hasta fines de 1988 en la versión 1.1, y se parecía un montón a Windows 2.0... por no decir que tenía la misma penosa interfaz. Luego vino la versión 1.2, la 1.3...

Pero nuevamente entre medias, en 1990, Microsoft había lanzado Windows 3.0. Y

**éste sí que fue un éxito tremendo**, de hecho éste fue el producto que catapultó a Microsoft a la posición que hoy tiene.

Porque no sólo IBM ofreció Windows 3.0 sobre MS/DOS (creo que por entonces sería la versión 4.0), sino también todo fabricante de ordenadores de la época.

En la ilustración podéis disfrutar de un típico escritorio de Windows 3.x, con su Reversi y todo, un escritorio que supongo que hace ya muchos años que no se ve en un ordenador de verdad.



Luego seguimos por aquí; como diría Raymond Burr (bueno, en realidad, el actor mexicano que le doblaba al español), en su papel de Ironside, el abogado paralítico que, tras Perry Mason, nos introdujo en las series televisivas de abogados en los años sesenta, “hagamos un *reseso*” para dar un vistazo a cómo estaba la tecnología del ordenador personal a principios de los noventa.

IBM había puesto a sus laboratorios a trabajar. Eran ingenieros realmente buenos, ya lo dije antes, y habían lanzado al mercado la gama **PS/2**, como antes comenté, a finales de 1987.

En la imagen, un PS/2 60 y un PS/2 80. Son prácticamente iguales por fuera, pero muy distintos por dentro: el primero llevaba un procesador Intel 80286, mientras el segundo llevaba el nuevo procesador 80386, bastante más rápido y con muchas mejoras sobre el 80286.



La base de estos ordenadores era su novedosa arquitectura de bus: *Microchannel Architecture*, o MCA. Era mucho mejor que la *arquitectura ISA*, que no era más que la arquitectura del PC/AT renombrada, que era el standard de todos los PC's de la época menos Apple, claro, siempre distinto. Se trataba de un avance enorme, pero es que además venía con un soporte BIOS completamente nuevo (ABIOS o Advanced BIOS), que solucionaba el trabajo en modo protegido y, por si fuera poco, introdujo también el standard gráfico **VGA**, algún tiempo después, XGA, cambió las conexiones de ratón y teclado a lo que, antes de la generalización del USB, ha sido durante muchos años el standard del mercado, conexiones que podéis ver en la ilustración.



Los componentes, por fin, eran de la calidad habitual de los productos de IBM (de la *IBM de aquellos años*, quiero decir): excelentes. Y con un soporte extraordinario.

Cualquier modelo de la gama PS/2 le daba cien vueltas a cualquier PC con la arquitectura ISA que usaban todo el resto de marcas, donde quizá, con suerte, podías obtener velocidades de transferencia de 5 Mb/s; con MCA obtenías tasas de 20, 40 o más Mb/s sin ningún problema.

Durante esa época yo tuve un PS/2 60, con un procesador 80286 igual que el AT al que sustituyó... y *volaba*, comparado con éste. No sé si decir que es el mejor ordenador que he usado en mi vida... lógicamente adaptado a las circunstancias y al momento, porque obviamente no se puede comparar en potencia al *monstruo* en el que escribo estas líneas veinte años después, aunque, para ser sincero, para escribirlas igual me hubiera servido mi provecto PS/2 60... Y en cuanto al IBM PS/2 80, que venía ya equipado con un procesador Intel 80386... era un *cañón*.

¿He dicho ya que *IBM se equivocó de medio a medio con su estrategia comercial respecto a su arquitectura microchannel*? Pues si no lo he dicho, lo digo ahora: metió la pata hasta el cuello. Volvió a malinterpretar el mercado, no dándose cuenta de que ya no estaba vendiendo tecnología sólo a las empresas, que no tenían mucho inconveniente en pagar más por tener una mejor calidad y garantía.

Efectivamente, el mercado del PC estaba girando rápidamente para convertirse en un *mercado de consumo*, donde **el cliente final es el consumidor individual**, que no sabe casi nada de arquitecturas, multithreadings ni BIOS *atómicas*, pero que **de dólares sabe un**

**rato...** o de pesetas, o de francos, o pesos, o marcos... no, de euros, no, nadie sabía nada, que aún no los habían inventado.

IBM tuvo la oportunidad de convertir la arquitectura MCA en el nuevo standard de mercado, pero no lo consiguió, y esta vez por avaricia. Su política no fue guardarse para sí su innovación, no, pues incluso en la prepotente IBM sabían que eso era comercialmente malo. Fue más sutil que eso: sí que ofrecieron licenciar su nueva arquitectura, patentada y protegida por royalties, a aquellos fabricantes que desearan implementarla en sus PC's, pero tenían que pagar una cantidad considerable por la transferencia de la tecnología (lógico), y además *pagar a IBM un buen royalty por cada máquina que vendieran*.

Como por aquí no transigió prácticamente nadie, el mercado de los PC's se fracturó en dos grupos irreconciliables. Por un lado IBM y su maravillosa (y cara) tecnología, la poderosa maquinaria de marketing del líder indiscutible y el apoyo de sus incondicionales, casi todos en las empresas. Y por el otro, todo el resto de fabricantes, excluyendo a Apple, que, como de costumbre, iba *a su bola* apoyado en la enorme superioridad del sistema gráfico de sus ordenadores. Estos fabricantes eran, uno a uno, mucho más pequeños que IBM, pero juntos... *juntos eran muy grandes*.

Y entonces hubo una especie de *conjura espontánea* (o no *tan* espontánea) entre todos ellos: **¡Todos contra IBM!** Tenían sus razones, desde luego: *si no ganaban esta batalla, desaparecían*. Así de simple. Y se pusieron manos a la obra, por un lado, por la vía del precio, ¡una vía efectiva, pardiez!; por otro, respondiendo con su propia arquitectura competidora (peor, vale, pero competidora): la EISA, anunciada y adoptada simultáneamente por los Nueve principales fabricantes de clónicos que competían con IBM, el llamado *Gang of Nine*, liderado por Compaq; y finalmente, por la vía del marketing.

La idea era desacreditar la arquitectura MCA de IBM, y los argumentos para hacerlo eran, más o menos: Que era propietaria y caerías en las garras de IBM para siempre jamás... Que tampoco era para tanto, total, para las aplicaciones que hay ahora, para qué quieres más... Que la competencia era buena y el monopolio era malo, e IBM quería el monopolio (*como todos*, ¡no te fastidia!, pero bueno)...

En el mercado empresarial, el ataque combinado de decenas de comerciales de compañías diferentes repitiendo con mínimas diferencias el mismo mensaje tuvo, al principio, no demasiado éxito, pero poco a poco fue calando. Y el mercado doméstico sí que escuchó todas estas razones, aunque, sobre todo, comparaba los precios... y decidía comprar lo barato, aunque fuera peor.

Como consecuencia, IBM estaba perdiendo claramente el mercado doméstico, que tampoco era tan grande todavía: a principios de los 90 tampoco había tanto particular con un PC en casa; sólo a partir de mediados o finales de los noventa se convirtió en normal tener un PC en el salón o en el despacho, siempre que tuvieras un despacho, claro.

Y entonces, curiosamente, profesionales que en su empresa habían confiado y confiaban ciegamente en la tecnología de IBM se compraban un clónico baratito para casa... y como la cosa, en realidad, funcionaba bastante bien, fueron paulatinamente perdiendo el *miedo al clónico*.

La **estrategia FUD**, sembrar en el mercado duda, incertidumbre y miedo (del inglés *Fear, Uncertainty and Doubt*) sobre los competidores que IBM había practicado extensivamente en el pasado (*"Si le compras un Sistema a ese pelagatos competidor mío, no te funcionará nada, perderás el dinero y el empleo, te dejará en la estacada, te quedarás calvo y no volverás a ligar en tu vida"*; cosas parecidas se las he oído yo a

comerciales de IBM toda la vida... y a los de otras marcas, también), ahora se volvió de golpe contra ellos, porque el que más, el que menos, respondía: “Pero... ¿cómo que los PC’s de *Fulano* son malos, malísimos de la muerte? Pues mi hijo tiene uno en casa y no veas cómo va allí el *Doom*, así que... menos lobos, Caperucita”.

Varios años más tarde, hacia 1993 ó 1994, no quedó más remedio que modificar de una vez la arquitectura ISA y su secuela, la EISA, porque hasta sus acérrimos defensores (porque comercialmente no tenían otro remedio), tuvieron que aceptar que era una *patata* y construir por fin la arquitectura PCI, tan normal en los ordenadores actuales.

Además, todos estos fabricantes de clónicos dotaban a sus equipos con un MS/DOS de Microsoft. Que cobraba las licencias y no pasaba de ellas ni un céntimo a IBM, ya que era Microsoft el propietario del sistema. Ojo, no menospreciemos a estos nueve fabricantes. Vale que eran más pequeños que IBM, pero entre ellos, además de Compaq, estaban Hewlett-Packard, Epson, NEC, Olivetti, etc. Además, había otros muchos fabricantes que hacían también PC’s, como Philips, Siemens, etc. De hecho, sólo NCR, entre los importantes, licenció la arquitectura MCA, con escaso éxito, además. O sea, *mercado dixit: MCA = IBM*.

Luego, a partir de 1990, todos estos fabricantes también ofrecieron Windows 3.0 (y luego 3.1, 3.11... y a partir de 1995, Windows 95 y sus secuelas, pero ésa es otra historia y será contada en otro momento).

Resumiendo, hacia 1991-1992, el panorama del mundo del PC era el siguiente:

**IBM no había ganado la batalla del hardware.** Vendía muchos excelentes PS/2, realmente muchos, pero la competencia conjunta de todos sus competidores estaba dando la vuelta a la tortilla: ya no era IBM quien marcaba las tendencias, por mucho que su *microchannel* fuera mucho mejor que ISA y EISA juntas.

IBM ofrecía el producto del desarrollo conjunto con Microsoft: el Sistema Operativo OS/2, por entonces ya en la versión 1.2 ó 1.3, que, sin duda, era sensiblemente mejor que MS/DOS con o sin Windows. Pero Microsoft había seguido su propio camino, desarrollando Windows a la vez que OS/2 y en algún momento de fines de los ochenta comenzó a desarrollar un nuevo (y esta vez, *a solas*, sin ninguna colaboración de IBM) Sistema Operativo, inicialmente pensado para ser un *OS/2 versión 2*.

En cuanto a Microsoft, seguía contando con IBM. Pero el éxito sin precedentes de su Windows 3.0 (sobre MS/DOS) hizo a Microsoft replantearse su relación con el gigante de entonces. Porque **MS/DOS con Windows estaba en la práctica totalidad de ordenadores que se vendían**, cualquiera que fuera su fabricante, mientras OS/2 sólo estaba, opcionalmente, en los equipos vendidos por IBM, que, muy a su pesar, lo que más vendía con diferencia seguía siendo esa *castaña* de MS/DOS, el ubicuo, omnipresente MS/DOS con Windows 3.0 o 3.1.



Podéis observar en la imagen uno de esos IBM PC's ejecutando el MS/DOS, con su características letras de color verde sobre fondo oscuro; de hecho, acaba de hacer un "CHKDSK" para comprobar que el disco no esté dañado, cosa bastante normal en la época. Anda que no he trabajado yo con uno de estos...

Sin duda alguna, OS/2 funcionaba perfectamente en cualquier ordenador de cualquier marca, pero sin embargo comenzó a ser *evidente* para todo el mundo que OS/2 sólo corría en hardware de IBM, aunque eso fuera falso de toda falsedad.

Y entonces a la fiesta se sumaron multitud de fabricantes de software, de dispositivos externos, como impresoras, o placas... todos, prácticamente todos ellos escribieron su software, sus aplicaciones, sus drivers, etc. en primer lugar para el Sistema que, con diferencia, más cuota de mercado tenía: MS/DOS y Windows. Y sólo después migraban sus productos a OS/2 y al resto de posibles sistemas, como SCO UNIX, etc. Después...o nunca.

Quizás el hecho de que el API para programar aplicaciones OS/2 lo vendiera IBM nada menos que a unos 4.000 dólares por licencia tuvo algo que ver... porque a las empresas desarrolladoras de productos les costaba 4.000 *dolores* pagar tal suma y, siempre que podían, lo evitaban, que a nadie le gusta padecer, y menos en la cuenta de resultados.

Comenzó, en fin, a ser patente para nosotros, los clientes, la cada vez mayor falta de software para usar en el excelente pero huérfano OS/2. Eso fue lo que decantó definitivamente la competición hacia el lado de Microsoft.

Desde el punto de vista de Microsoft, su postura es lógica, sin duda alguna: visto el éxito de MS/DOS y de Windows, lo lógico para ellos era concentrar esfuerzos en dar soporte y mejorar ambos, cuidando su mercado y su enorme base instalada y en paralelo escribir un Sistema Operativo moderno para las futuras aplicaciones o para correr en los servidores, casi inexistentes en esa época pero que la paulatina mejora en las redes de comunicación comenzaba a hacer factibles.

En algún momento de aquellos años nos planteamos seriamente en mi empresa, y sé que otras muchas empresas hicieron lo mismo, si cambiar nuestros Sistemas Operativos de PC, abandonando MS/DOS y Windows y adoptar en su lugar OS/2. Éramos una empresa muy "azul", muy fieles a IBM, que la verdad es que nunca nos había dejado en la estacada hasta entonces, la presión de los comerciales de IBM fue intensa, el precio ofrecido era muy competitivo... y sin embargo, decidimos seguir usando MS/DOS y Windows y, más adelante, Windows NT, 95, etc. La evidente escasez de software de terceros migrado a OS/2 fue absolutamente determinante en la decisión.

En cierto momento, sobre 1990, **IBM y Microsoft decidieron dedicarse cada una a lo suyo**: IBM a desarrollar y mejorar OS/2 y Microsoft a su MS/DOS, su Windows y el nuevo sistema que estaba escribiendo, lo que más tarde sería Windows NT. Y aunque el acuerdo garantizaba a cada uno acceso al código del otro, lo cierto es que las relaciones se agriaron muchísimo, y muy rápidamente.

En 1992 IBM, ya en solitario, lanzó al mercado su OS/2 versión 2. Era un Sistema Operativo extraordinario, lo conocí bien y puedo asegurarlo. **Fracasó**. Bueno, no fracasó de golpe, vendió licencias, casi siempre en entornos empresariales, nunca al particular, estuvo vigente muchos años, incluso sus herederos lo siguen estando hoy... pero si lo comparamos con Windows... pues eso, que fracasó. Completamente.





Hasta septiembre de 1993 no estuvo Windows NT 3.1 en el mercado. Lo llamaron “3.1” en lugar de “1.0”, que hubiera sido lo normal para la primera versión operativa, para hacer coincidir su número de versión con la de Windows 3.1, que era la versión en vigor en ese momento. Repito: *un maestro del marketing*, el amigo Gates.

Sí, Windows NT salió al mercado en 1993, ya a finales, para ser exacto. Pero estaba *anunciado* desde mucho tiempo antes, bastante más de un año. Siempre se retrasaba porque había que hacer *no-sé-qué nueva prueba adicional* para garantizar el súper-performance, o la súper-compatibilidad con algo, o que no tenía fallos... Y por medio, el *run-run* incesante, el *FUD* por parte de todos los fabricantes de clónicos competidores de IBM, sin excepción, que no daba tregua. La presión mediática era realmente fuerte... mucho.

Se decía que OS/2 v2 necesitaba ¡2 Mb! para funcionar, nada menos, pero ¡*qué barbaridad, oiga!* Que sólo funcionaba en hardware de IBM y, por tanto, si comprabas OS/2 quedabas ligado a los equipos de IBM para siempre. Que los ingenieros buenos, pero *buenos de verdad*, eran los de Microsoft, mientras que los de IBM sabrían muchísimo de mainframe, pero de PC's, nada de nada. Que la compatibilidad con aplicaciones MS/DOS Windows no funcionaría nunca... Y todo así.

#### **FUD. Todo FUD.**

Bueno, no, la primera afirmación, no lo era. Es totalmente cierto que OS/2, versión 2, necesitaba sus buenos 2 Mb de memoria para ir bien...

Claro que Windows NT, una vez que salió al mercado, **necesitaba cuatro**... Pero, por algún sorprendente motivo, en 1993 eso no importaba nada cuando en 1992 era un problema gigantesco... me recuerda a esos políticos que piden la dimisión inmediata, el encarcelamiento y la picota para cualquier cargo de *otro partido* cuando es pillado robando... pero cuando al que le pillan robando es a alguien de *su propio partido*, entonces es cuando se acuerda de eso de “todos somos inocentes, mientras no se demuestre lo contrario”...

Del resto... OS/2 v2 funcionaba en todo tipo de hardware, no sólo en IBM; los ingenieros de IBM eran buenísimos, aunque también lo eran los de Microsoft, que habían fichado a Dave Cutler, el creador de VMS en Digital para liderar el proyecto de NT; y, por fin, yo he visto Windows 3.1 corriendo en una ventana de OS/2 tan ricamente...

Tras toda esa *letal campaña anti-OS/2*, lanzada desde tantos frentes a la vez, comenzó a ser palpable en el ambiente una sensación de “*esto está muerto, difunto cadáver*” cuando se hablaba de OS/2, que impregnó al mercado y condicionó muchas

decisiones de compra en empresas... y no digamos entre los particulares.

OS/2 no tuvo nada que hacer contra todo este ambiente; IBM no sabía realmente cómo resucitar a un software excelente, mejor que prácticamente toda su competencia, pero que *nadie quería*... No se me ocurre nada mejor para que os hagáis una idea de cómo estaba el patio aquellos años que contaros un ejemplo, un *sucedido* que cuento de primera mano, porque *lo vi con mis propios ojos y lo oí con mis propios oídos*.

**Convención Europea de Gartner Group**, noviembre de 1992, Hotel Loew's, Montecarlo (Mónaco). Dos mil y pico profesionales de toda Europa, quizá más, asistiendo para oír a los *gurús gartnerianos* evangelizarnos sobre el futuro de la tecnología (y poco menos que de nuestras vidas), por el módico de precio de *no-me-acuerdo-cuánto-pero-una-pasta-gansa* por cabeza...

Conferencia sobre el futuro de los Sistemas Operativos de PC, protagonizado por uno de los más prestigiosos *analistas de referencia* de Gartner, de cuyo nombre ni me acuerdo ni quiero acordarme.

Máxima expectación en la sala (enorme), llena hasta los topes, quizá mil quinientas personas, quizá más. Traducción simultánea a cinco o seis idiomas, entre ellos español, pero el traductor era tan penoso que yo, al menos, escuché la conferencia directamente en inglés. Mi inglés entonces era malo (*eeh, bueno, no es que sea mucho mejor ahora*), pero sí suficiente como para poder seguir perfectamente la conferencia.

Después de casi una hora contándonos cosas similares a las que acabo yo de contaros, pero revestidas de la *auctoritas* que otorga el *hablar correctamente en inglés*, y otras más, todas aproximadamente en la misma dirección, para acabar su charla va y nos cuenta un chiste.

Sí, sí, *un chiste*, lo prometo. Debía ser alguna técnica americana de *presentación eficaz* o algo así. Reproduzco a continuación el dichoso chiste lo más textualmente que sé:

«Están reunidos el Presidente de los EEUU, el Presidente de Rusia y el Presidente de IBM, deliberando sobre el futuro del mundo y tal.

»De pronto se oye un gran trueno y se aparece el mismísimo Dios en su Infinita Majestad y con tonante voz dice, dirigiéndose al Presidente de EEUU:

»- *Como eres el hombre más poderoso del planeta, te comunico que dentro de una semana se acabará el mundo...* Y desaparece en medio de una nube, con rayos y centellas y ángeles trompeteros.

»La reunión se da por terminada y cada cuál se vuelve rápidamente a la Casa Blanca, al Kremlin y a las Oficinas Centrales de IBM, respectivamente, y cada cuál reúne a su Consejo para darle la nueva.

»El Presidente de los EEUU expone a su Consejo de Secretarios de Estado:

»- *He de comunicaros dos noticias, una, buena, y la otra, mala. La **buena** es que Dios ha hablado al Presidente de los Estados Unidos. La **mala** es que el mundo se acaba en una semana.*

»Por su parte el Presidente ruso dice a su Consejo de Ministros:

»- *He de comunicaros dos noticias, una, mala y la otra, peor. La **mala** es que Dios ha hablado al Presidente de Estados Unidos. La **peor** es que el mundo se acaba en una semana.*

»Y, por fin, el presidente de IBM explica a su Consejo de Administración:

»- *He de comunicaros dos noticias, una, buena, y la otra, mejor. La **buena** es que*



*Dios ha hablado al Presidente de Estados Unidos. La **mejor** es que dentro de una semana a nadie le importará lo que vamos a hacer con OS/2...»*

Fin del chiste.

Prometo haberlo oído exactamente así y en tales circunstancias, así que podéis haceros una idea de cómo estaba el percal en 1992-93... La verdad es que IBM perdió completamente la batalla del software en el mundo del PC y OS/2 quedó como una reliquia, una reliquia vistosísima, magnífica, pero reliquia al fin, como el Video 2000 de Philips que comentaba hace algunos capítulos, caro, con poco soporte y casi circunscrito, a su pesar, a correr en máquinas de IBM.

Microsoft se forró... y aprendió. Entró en el mercado de las suites ofimáticas, desplazando a Lotus, que con su Lotus 1-2-3, su Amipro, su Freelance Graphics y su Approach dominaba de largo este mercado, donde había además otros muchos competidores como Wordstar, Wordperfect, DBase, etc. IBM intentó recuperar cuota de mercado comprando a Lotus, fundamentalmente por su Lotus Notes, el correo electrónico interno de referencia aún hoy en muchísimas instalaciones, pero la última versión de la suite de Lotus, de nombre Lotus Smartsuite, data de 1999... y entonces IBM prácticamente la regalaba: no tenía ningún futuro, pues Microsoft Office se la había *comido por las patas*.

Entró Microsoft también en el mercado de los navegadores, desplazando al dominador Netscape, historia ésta muy bien conocida debido a las grandes repercusiones jurídicas y mediáticas que tuvo en su momento. Se metió también en el mundo de las Bases de Datos Relacionales con su SQL Server, consiguiendo con el tiempo ser el líder indiscutible en Bases de Datos en el entorno Windows, desplazando por completo a Oracle, Sybase, DB2 y al resto, que entre todos ellos dominaban inicialmente ese mercado...

Cabalgando, por fin, sobre la enorme influencia de sus omnipresentes versiones del *Sistema Operativo de las Ventanas*, llegó Microsoft a ser lo que es hoy y a hacer de su fundador durante muchos años el hombre más rico del Universo... pero ésa es otra historia y será contada en otro momento.

Porque los problemas para IBM no sólo venían de sus cuitas con los PC's y sus Sistemas Operativos: también estaba comenzando a tener serios problemas en áreas de la tecnología donde tradicionalmente había gozado de una gran tranquilidad... y rentabilidad. Y a estos acontecimientos, que, no se os olvide, ocurrían de forma simultánea a todo lo relatado aquí, está dedicado el siguiente capítulo.

*Moviditos, los noventa, ¿verdad?*

## 16 - Los años noventa (II): La ascensión de los minis

Hemos visto mi como siempre personal repaso a los hechos que acontecieron durante finales de los ochenta y la primera parte de los noventa, hechos que acabaron con el dominio de IBM en el lucrativo mundo del PC y *cambiando el foco* desde el hardware al software: a partir de esos años, las compañías que fabricaban hardware dejaron de ser las dominadoras del mercado, lugar en el que les sucedieron las compañías fabricantes de Sistemas Operativos y Suites Informáticas.

Ahora me centraré en las convulsiones similares que acontecieron esa década en el mundo de los llamados “miniordenadores”, cuya consecuencia final fue, por un lado, la propia pérdida del concepto de “*miniordenador*” y por otro, igual que en el mundo del PC, en el cambio de foco de la rentabilidad desde el hardware, donde siempre había estado, hacia el software, situación que continúa plenamente vigente en nuestros días.

Una vez más advierto contaré todo esto tal como yo lo viví o, al menos, como lo recuerdo, así que puede haber multitud de errores e inexactitudes, por los que pido disculpas anticipadamente.

Además, debéis tener en cuenta que todo lo que os contaré aquí sucedía simultáneamente a la guerra de los PC's de que os hablé en el capítulo anterior y a muchas otras cosas que ocurrían en todos los frentes restantes... Fueron años divertidos, los 90 del siglo pasado. Y estresantes, ya lo creo. Muy estresantes.

A principios de los noventa **IBM** era el líder mundial de ventas, aunque veremos luego que, aunque vendía mucho, no ganaba *tanto*. Con sus mainframes y el software básico para ellos dominaba sin ambages el lucrativo mundo de los grandes ordenadores empresariales, situación que, por cierto, no ha cambiado prácticamente nada hasta nuestros días, donde esa División de Mainframes sigue siendo de las más rentables de IBM en la actualidad. No digo yo que sea la que más vende, que no lo es, sino que es muy rentable, aunque habría que saber dónde computan en los últimos años los contables de IBM los ingresos por software y servicios relacionados con los mainframes. Como lo que ahora *mola* es que el núcleo del negocio de las empresas de tecnología sean los servicios y el software y no tanto el *hierro*, igual llevan todos esos ingresos a las Divisiones de Software y Servicios para salir más *guapos* en la foto. *Garabos* contables, todo el mundo los hace para contentar a los analistas financieros y que te pongan en esa categoría tan deseada de *Sobreponderar*... últimamente bastante desierta, pero algún día cambiarán las cosas. ¡Espero!

El Sistema ES/9000, lanzado al mercado en 1990, tenía ya muchas de las características de los modernos mainframes y fue la base de la oferta de mainframes de IBM durante los años noventa; ya entonces casi no tenía IBM competencia en este mercado más allá de los PCM's, los sistemas clónicos de los mainframes de los que hablé en el capítulo correspondiente y, conforme pasaba la década, incluso esa competencia acabó por desaparecer finalmente.

Pero ya dije en el mismo capítulo antes citado que, en esta área, el problema de IBM venía del hecho de que la práctica totalidad de grandes empresas e instituciones mundiales tenían ya su mainframe o varios de ellos, pagaban religiosamente sus cuotas anuales por el

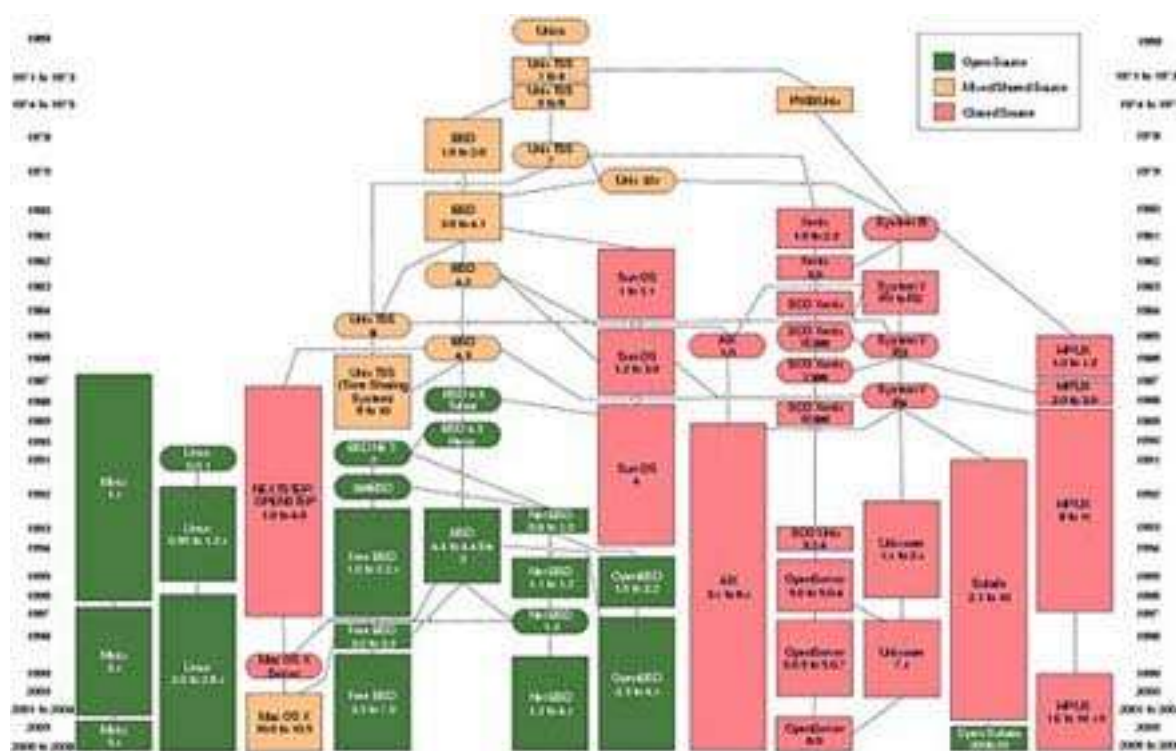
software, ampliaban sus equipos o los cambiaban por otros más modernos y potentes... **pero apenas conseguía nuevos clientes.** Me parece a mí que eso de no aumentar la base de clientes debe resultar malo para cualquier compañía. Y, desde luego, lo era ya para la IBM de fines de los ochenta y los noventa.

En el mundo de los ordenadores medianos, los “minis”, IBM tenía sobre todo bien amarrado el mercado de empresas comerciales medianas, con su gama de ordenadores AS400, sucesora de los System/3x.

Si todos estos sistemas vendían bastante era, aparte de por su innegable calidad, por la cantidad de aplicaciones empresariales existentes para estos sistemas en esos años, que permitían a esas empresas sacar partido al sistema rápidamente sin grandes inversiones en Desarrollo.

Pero aquí sí que tenía IBM una fortísima competencia: multitud de fabricantes tenían excelentes sistemas, la mayoría basados en algún tipo de UNIX debidamente *mejorado* (y debidamente patentado) por la marca: HP/UX, en el caso de Hewlett-Packard, Digital UNIX en caso de Digital (aún vendía Digital sus veteranos sistemas VAX con el Sistema VMS, pero pronto dejó de hacerlo), Solaris, en el caso de Sun, UNIX System/5, por parte de NCR y otros y un largo etcétera, incluyendo también SCO UNIX para PC, con poquísimo mercado aún, pero creciendo paulatinamente.

En la ilustración os muestro un diagrama, cortesía de la Wikipedia, de las decenas y decenas de variantes de UNIX y cómo han ido evolucionando de unos a otros y mutando continuamente... ¡ni que fueran *Pokemon*!



Todos estos sistemas *pequeños* habían estado tradicionalmente dirigidos sobre todo a entornos de ingeniería o científicos, o bien para entornos técnicos, como telefonía y

comunicaciones, por ejemplo, pero muy poco hacia empresas *comerciales*: Bancos, Aseguradoras, Minoristas, etc., o sea, las que de verdad disponían de cantidades ingentes de dinero para gastar.

Pero esto, a finales de los ochenta y principios de los noventa, estaba cambiando rápidamente: por una parte, las máquinas eran cada vez más potentes, por mucho que, como prácticamente la totalidad seguían siendo de un solo procesador, no podían competir ni de lejos con un mainframe en esto y, por otro, el software había mejorado ostensiblemente, tanto el propio Sistema Operativo, como el resto de software de base, especialmente los Gestores de Bases de Datos.

Tanto Oracle como Informix, Sybase, Ingres y otras Bases de Datos Relacionales estaban disponibles para los diferentes olores, sabores y colores de UNIX, y esto permitió aumentar la oferta de estos sistemas para esas aplicaciones empresariales y comerciales que hasta entonces habían sido prácticamente coto vedado de los mainframes.

**Hewlett-Packard** tenía en el mercado desde mediados de los setenta su serie HP3000, con el Sistema Operativo MPE y una más que probada fiabilidad que le había generado suculentas ventas durante muchos años. De hecho, la serie HP3000 sólo se dejó de vender en una fecha tan tardía como 2003. Pero tras la adquisición de Apollo Computers en 1989 y la inclusión de sus nuevos procesadores PA/RISC, potenció la serie de estaciones de trabajo HP9000, equipada con HP/UX, entrando como una locomotora en el mercado de servidores y convirtiéndola pronto en su buque insignia.

Si añadimos a esto que Hewlett-Packard dominaba de largo el mercado de impresoras para PC, sobre todo las impresoras láser con su gama LaserJet, y que era uno de los fabricantes de clónicos de PC con mayor actividad comercial, podemos añadir al cóctel de su buena gama de productos una respetable posición empresarial.

Estaba ya considerada HP como “*una empresa seria*” por parte de sus potenciales clientes; iremos viendo que esta *respetabilidad* resultó un factor importante de éxito en lo que iba a venir.

Varias compañías fundadas a principios de los ochenta estaban también creciendo a marchas forzadas, fabricando y vendiendo estaciones de trabajo y servidores de gran calidad basados en UNIX, cada una en *su UNIX*, naturalmente, hasta ahí podíamos llegar: todo el mundo intentaba crear su propio mercado cautivo.

Una de ellas era **Silicon Graphics**, especializada en costosas workstations, sobre todo para el mundo del diseño gráfico, el vídeo y los gráficos avanzados en entorno técnico, aplicaciones todas ellas que requerían una potencia de hardware inalcanzable para los PC's o las workstations *normales* de la época, las más orientadas al propósito general.

En este mercado del tratamiento de vídeo tuvieron los de Silicon Graphics una posición prominente hasta finales de los noventa. Aunque la mayor parte de su facturación la hacía con el Gobierno estadounidense, la base de su publicidad era asegurar que los estudios de Hollywood eran sus mejores clientes, seguramente porque buena parte de sus proyectos gubernamentales eran confidenciales. Sin embargo, cuando la evolución tanto del software como del hardware hizo que hasta los PC's más *normalitos* tuvieran muy buenas capacidades para tratar vídeo... pues lo pasaron mal. Realmente mal. Tanto, como que SGI se declaró en bancarrota en 2006...

Por su parte **Sun**, que había estado exclusivamente dedicada a las workstations

como la Sun 150 de 1983, pero basadas a partir de 1987 en su propio procesador SPARC, entró también en el mercado de servidores a partir de 1992 o 1993, ofreciendo máquinas de procesamiento simétrico (SMP) con buenos rendimientos y mejor precio, que usaban como software... su propia versión de UNIX, claro: Solaris.

Sun fue ampliando paulatinamente su gama de servidores, cada vez más potentes, más fiables y con mejor precio, y fue adquiriendo una buena posición en este mercado, que mejoró ostensiblemente a raíz de su fervorosa adhesión al mundo naciente de internet, sobre todo a raíz de la publicación de *su* Java en 1995... Y consiguió muchas ventas, mucha rentabilidad y se convirtió en uno de los líderes de fines de los noventa... y de los que más sufrió a raíz del estallido de la *burbuja puntocom*, en 2001. Tras unos años complicados fue adquirida en 2009 por... Oracle. Nada menos que por *Oracle*, una compañía dedicada en exclusiva desde su nacimiento al software y los servicios... Parece que el motivo de que Oracle se interesara por Sun hasta comprarla finalmente es mucho más por su floreciente Java que por sus máquinas... pero ésta es otra historia y será contada en otro momento.

Volviendo a los noventa del Siglo pasado, en 1994 Sun había comprado a Thinking Machines, una compañía especializada en la fabricación de supercomputadores vectoriales. Entre las compañías especializadas en proceso vectorial destacaban, además de la propia Thinking Machines, CRAY, que fue comprada aquellos años por Silicon Graphics, luego separada, revendida otra vez, y que continúa existiendo en la actualidad, y Convex, también comprada en su día por Hewlett-Packard.

Incorporando en parte la tecnología proveniente de Thinking Machines a sus máquinas, Sun pudo poner en el mercado en la segunda mitad de la década de los noventa servidores SMP con Solaris, con decenas de procesadores y una ingente capacidad de proceso. Y vendió muchas unidades... mientras duró.

**Digital Equipment**, otro de los grandes del mercado *mini* del momento, estaba, en cambio, pasándolo mal.

En 1960, nada menos, inventaron el concepto de miniordenador: los de Digital fueron los primeros con el **PDP-1**, aunque, viendo su foto, a duras penas podemos pensar que *eso* fuera un *mini* ordenador... ¡Cómo serían en la década de los sesenta los ordenadores-a-secas...!



Digital había dominado el mercado de los minis, sobre todo a partir del lanzamiento del PDP-8, en 1965 y durante toda la década de los setenta y ochenta, gracias a sus series PDP y VAX. Pero no había encajado bien el cambio tecnológico y comercial sufrido por la informática a partir de la llegada del PC en los años ochenta.

Tras intentar encontrar su hueco en este mercado sin éxito, su respuesta final fue la creación del nuevo chip Alpha, lanzado en 1992, con arquitectura RISC de 64 bits y una excelente calidad... y, consecuentemente, un elevado precio. El mercado no estaba preparado para un procesador tan novedoso (léase: *potente y caro*), y además hubo que modificar los Sistemas Operativos para que usasen este avanzado chip: tanto VMS, como Digital UNIX y Windows NT debieron sufrir una costosa adaptación.

Cuando Digital lanzó por fin la gama AlphaServer, en 1994... ya era tarde. Pocos años después fue comprada por uno de sus competidores, uno impensable hasta hacía muy poco tiempo: **Compaq**, que había conseguido su solvente posición fabricando y vendiendo PC's. ¡*El acabóose!* (imítese aquí la grave voz del desaparecido Fernando Fernán Gómez, por ejemplo en "El viaje a ninguna parte"), pensamos muchos... Claro que pocos meses antes Compaq había adquirido también al creador y *alma mater* de la tolerancia a fallos, Tandem Computers, así que tampoco era *tan* raro, ¿no?

La propia Compaq pagó en sus carnes los excesos de la "*burbuja puntocom*" pocos años después y fue a su vez adquirida por Hewlett-Packard... pero ésa es otra historia y será contada en otro momento.

Había muchos más fabricantes de Sistemas UNIX, como Philips, Unisys, NCR, Nixdorf y un largo etcétera, cada cuál defendiendo su pequeña parcela con uñas y dientes e intentando arañar un pedacito de la parcela del vecino...

Y, por fin, la propia **IBM** se había dado cuenta, a finales de los ochenta, del creciente potencial de este tipo de sistemas, los hasta hacía poco denostados (*por IBM*, quiero decir) "miniordenadores", y decidió entrar también en el mercado de los sistemas pequeños basados en UNIX.

Y ¡cómo lo hizo!... a lo grande, como IBM solía hacer las cosas: sacó al mercado la gama **RS/6000**, que venía equipada con novísimos procesadores RISC: los POWER, de diseño propio, como todo el resto de componentes, siempre con tecnología propietaria y protegida por cientos de patentes, o sea, a la *manera de IBM*. Como Sistema Operativo, estas máquinas llevaban un UNIX, sí... pero, cómo no, uno con patente de IBM: AIX, para acabar de liar la madeja de UNIX y más UNIX que había en el mercado... Y completó IBM la salida al mercado de la gama RS/6000 con el lanzamiento de una gran cantidad de software básico para estas nuevas máquinas, entre ellos DB2/6000, la versión de su Base de Datos Relacional para entornos UNIX.

Sin embargo, para IBM no era éste un mercado conocido. Tuvieron que crear la tecnología, contratar o formar a sus técnicos, a sus comerciales, lanzar una estrategia de marketing, etc. En una palabra, necesitaron un tiempo para asentarse en el mercado y ser percibidos como un competidor fiable en el mundo UNIX, el mundo de los "Sistemas abiertos", que es como se autollamaban. Para agilizarlo, se asociaron a todas las asociaciones habidas y por haber, como la OSF, y a todos los estándares existentes en el mundo mundial, como POSIX (al menos, *de boquilla*, como casi todos)... todo ello para crearse una reputación de "*apertura*" que el mercado, es decir, nosotros los clientes, estábamos empezando a valorar, aunque **a duras penas se encontrara un sistema "abierto" de verdad esos años...**

Esta última afirmación la hago con conocimiento de causa, pues en aquel tiempo diseñamos y escribimos en mi empresa una aplicación magnífica sobre un servidor HP9000 con HP/UX, *graciosa y muy bien parida*, pero no excesivamente complicada, que funcionaba muy bien. Luego, por causas diversas que no vienen al caso, hubo que migrarla en primer lugar a un servidor Digital con Digital UNIX, luego a un RS/6000 de IBM con AIX, al poco tiempo a un Sun con Solaris... y cada migración fue un dolor de cabeza, tanto que casi tardábamos más en migrar la aplicación de UNIX a UNIX que lo que habíamos tardado en escribirla. En realidad estoy usando el plural mayestático, porque quien escribió la aplicación fue *mi equipo*; yo me limitaba exclusivamente a *planificar, exigir, premiar y castigar*, je, je.

¡Me van a contar a mí lo de los **Sistemas Abiertos** de los noventa...! No sé qué pasará ahora con los Linux, no estoy en ese negocio... pero lo que oigo es que pasan cosas parecidas, aunque ciertamente con menor intensidad que con los dichosos UNIX de los noventa... Algo se habrá aprendido en quince años o veinte años, digo yo.

El caso es que sobre los años 91-92 del pasado siglo había una considerable oferta de sistemas UNIX de diferentes fabricantes, pretendidamente intercambiables entre sí, gracias a que UNIX *aseguraba* (o *así*) la portabilidad de aplicaciones entre uno y otro, con cada vez mayor capacidad de proceso y almacenamiento y un precio realmente competitivo, sobre todo si lo comparamos con el que tenía un mainframe tradicional en la época.

Poco a poco fueron canibalizando, en parte, al menos, las hasta entonces exclusivas señas de identidad del *mundo mainframe*, a saber: Fiabilidad, Potencia de Cálculo, Compatibilidad hacia delante, buen soporte, etc. Y con mucho menor coste *aparente*. Pero lo que pasaba es que las grandes empresas no tenían apenas Técnicos de Sistemas UNIX, de ninguna clase de UNIX, si es que tenían alguno. Ni Analistas, ni Programadores. Ni Operadores. Ninguno de ellos.

Así que, además de comprar las máquinas y su software básico, había que comprar el resto de software necesario, de comunicaciones, formar (o fichar) a los técnicos, a los usuarios... Además, resulta que los sistemas UNIX, al menos los de la época, se volvían inestables cuando la CPU llegaba más allá del 70-80% de su capacidad... pero la publicidad no decía nada de todo esto, claro está.

En definitiva, vale que tanto las máquinas como el software fueran sensiblemente más baratos que un buen mainframe, pero **una migración a este tipo de sistemas tenía unos elevados costes ocultos**... que nadie reconocía, naturalmente. Además, es posible que a las empresas, es decir, a los clientes finales, sí que les importaran los costes ocultos, pero, a las Consultoras, Compañías de Servicios Profesionales y a los propios fabricantes... *¡De ninguna manera!* Todo lo contrario, de hecho: vieron una oportunidad de oro de elevar sus ventas, sus márgenes, su facturación y, en definitiva, la retribución de sus comerciales, del Consejo y de los accionistas.

Así que se lanzaron todos a una, como si hubieran oído tambores de guerra, a la yugular de la durante tanto tiempo prepotente IBM, encarnada, fundamentalmente, en su hasta hacía muy poco omnipotente División de Mainframes.

Los argumentos que utilizaban eran, no hay que ser un genio para darse cuenta, *demoledores*:

1) Estaban promocionando **la nueva tecnología**, en contraposición con la *vieja tecnología* del mainframe. Que ésta fuera probada, potente, fiable y bien conocida no importaba nada. Más bien era al revés...

2) Con tantos competidores y una competición tan intensa entre ellos, **la tecnología iba a evolucionar rápidamente**, proporcionando previsiblemente grandes aumentos de rendimiento y sensibles bajadas de los precios en un futuro próximo.

3) **Precios que, además, ya eran más bajos** que los de la tecnología de los dinosaurios.

4) Como ya se vislumbraba el boom de internet en el horizonte (estamos en 1993-1994, más o menos), **se requerirían nuevos servidores... muchos servidores**, y no íbamos a cometer la estupidez de comprar nuevos y caros mainframes para tales cosas mundanas.

5) Y por si fuera poco, **¡ni siquiera se programaban en Cobol!**, esa vieja *cosa del Cretácico Medio*, sino en los modernos lenguajes orientados a objetos que nos iban a solucionar la vida en un pis-pas, porque en muy pocos años, nadie programaría ya nada, sino que tomaría objetos de aquí y de allá, los ensamblaría todos juntos, *et voilà*: su aplicación a su antojo en tres días... Vale, también se podía programar en Cobol, que había excelentes Coboles para UNIX... pero ¿a qué *neanderthal* se le ocurriría hacer tal cosa y no contratársela a su proveedor favorito de Servicios Profesionales para que la hiciera en Visual Basic, en C, en C++, o en cualquier otra cosa *de moda*?

Tenía yo una agenda que se publicó sobre el noventa o noventa y dos, que hacía previsiones sobre qué nuevas tecnologías aparecerían en los próximos cincuenta o sesenta años, qué sé yo... Decía cosas como: 2000: Se impone el coche eléctrico. 2011: Se consigue por fin la energía de fusión. 2028: Se establece la primera colonia humana en las lunas de Júpiter... y así todo.

Bueno, pues en la agenda, las previsiones para, por ejemplo, 1998, eran: **Abandono definitivo del Cobol**. Y en 2003 decía: **El Cobol es definitivamente sustituido por lenguajes más modernos**. Y en 2008: **Sustitución definitiva de Cobol como lenguaje de programación**... y así cada cinco años. 2048: **Los informáticos del mundo consiguen por fin eliminar completamente el lenguaje Cobol**... Y así con todo.

En el resto de predicciones que hizo para años que han pasado no dio ni una, ni parece que las que quedan por llegar vayan a cumplirse tampoco, pero en ésta del Cobol, acertó de pleno: *¡No pasan cinco años en los que no se abandone de una vez por todas el dichoso Cobol!*

Bueno, sólo quedaba, para redondear la estrategia de acoso y derribo al líder, a IBM, dotarla de un adecuado acompañamiento de marketing: había que acuñar un término efectivo, poderoso y muy, muy comercial, para que cualquiera de la profesión pudiera identificar sin ningún género de dudas el mensaje.

Finalmente, el término que resultó elegido fue...

### **Downsizing.**

El significado original de la palabreja es muy parecido al término hispánico de *ERE* (Expediente de Regulación de Empleo): Reducir costes y eliminar personal (despidiéndole y enviándole de cabeza al paro) para aumentar así la eficacia de la empresa.



# Will you stay competitive?



Downsizing a Mammoth Organization Isn't Enough.  
Before its operation was downsized, this group was one of the largest in the field.  
Survival in the 90's will take more than downsizing and cost cutting. It will take greater flexibility, imagination, productivity, planning—a dedication to making your products, and yourself—the best.

Como entonces estábamos atravesando la *consabida crisis* del 92-93, el término dichoso tuvo inmediata aceptación: a los oídos de los Directores Financieros, de Tecnología, de Recursos, etc., a todos ellos el mensaje les sonaba a música celestial: **gastarse mucho menos dinero por lo mismo** es un dulce que a nadie amarga...

La cruda realidad era que ni era *tanto menos dinero*, ni era por *lo mismo*. De hecho, en ocasiones era *más* dinero por *menos* capacidad...

Sí, cierto, pero... ¿y a quién le importaban esas menudencias a la hora de vender un buen Plan Estratégico, seguido de un millón de horas/hombre de recursos cualificados...?

Comenzaron a proliferar por doquier Conferencias y Charlas varias, por no hablar de artículos en revistas del gremio, en las que ponente tras ponente, escritor tras escritor, consultor tras consultor, todos ellos realmente cualificadísimos, nos *evangelizaban* una y otra vez sobre las indiscutibles ventajas del downsizing. Todo eran ventajas. Ningún inconveniente, *of course*. Y si hubiera alguno, *aquí estamos nosotros para solucionárselos*... ¡faltaría más!

Los mensajes comenzaban, indudablemente, a calar entre los clientes. Es difícil oír una y otra vez a uno tras otro prestigioso consultor explicándote lo bueno del downsizing, lo malo de permanecer anclado a una tecnología llamada a desaparecer en breve (sí, eso se repetía una y otra vez: los mainframes estaban condenados a desaparecer en no más de cinco años), la cantidad de dinero que estabas tirando literalmente día a día a la basura... y seguir en tus trece. Máxime cuando al Jefe del Jefe de tu Jefe también le cuentan las mismas cosas los Jefes de los Jefes de los Consultores que te lo cuentan a ti...

Una pregunta que oí muchas veces en aquellos años, era: "...y en vuestra Organización... ¿cuándo vais a tirar los mainframes de una vez y sustituirlos por una buena granja de servidores UNIX y PC's (peazo dinosaurios, que sois unos triceratops...)?" Lo que está entre paréntesis no lo decían, claro está, salvo que fueran muy, muy amigos, pero lo pensaban, todos lo pensaban...

Incidentalmente, Microsoft lanzó en 1993, por fin, su esperadísimo y mil veces retrasado Windows NT. Jaleado por todos los fabricantes de PC's (excepto, claro está, IBM, que aun así tuvo que ofrecer Windows NT también para sus sistemas PS/2), por los consultores, las casas de software independientes, etc., resultó un gran éxito, quizá no de ventas, al menos de forma inmediata, pero sí mediático, con lo que la carrera del OS/2 languideció definitivamente.

Además, y como es lógico ante una nueva oportunidad de negocio, los fabricantes de software de base (los de Bases de Datos, sobre todo) se lanzaron rápidamente a migrar

sus productos a Windows NT, aumentando los sistemas susceptibles de recibir sistemas “downsizingueados” (perdón por el palabro, no he podido resistirme) y, por ende, la subsiguiente presión adicional para IBM y para los fabricantes de clónicos de los mainframes. Estos últimos lo pasaron tan mal que, en la práctica, o se dedicaron a otras cosas o simplemente desaparecieron en pocos años.

De todos modos, y a pesar de su indiscutible liderazgo tecnológico y en ventas, IBM lo estaba pasando mal también. Vendía más que nadie, sí, pero una vez hechas las cuentas... la de resultados, o sea, la única que de verdad importa, terminaba en números rojos, y cada vez más rojos hasta que en el ejercicio fiscal 1992 anunció pérdidas de nada menos que 8.100 millones de dólares de la época, las mayores jamás registradas hasta entonces por ninguna compañía de todo el mundo... aunque dadas las cifras que se manejan ahora en nuestra *crisis cotidiana de cada año*, estas cifras nos parecen ridículas. IBM siempre hacía todo a lo grande, como podéis ver. Incluso perder dinero.

Había muchas causas: la División de Mainframes era rentable pero estaba sobredimensionada (recordad que quedaban pocas empresas susceptibles de comprar un mainframe que no lo tuvieran ya); la anterior mina de oro de la División de Sistemas Personales, los PC's, vaya, sufría una fortísima competencia que había erosionado sus márgenes hasta comérselos; la División de Sistemas Generales, la que vendía AS/400, sufría una fuerte competencia de los fabricantes de minis... entre los que estaba la propia División de Sistemas UNIX de IBM, con sus novedosos RS/6000, que todavía no se habían consolidado en el mercado UNIX. Y además de todas estas causas tecnológicas, había también causas financiero/contables, debidas a un cambio de estrategia en la venta o alquiler de equipos, que no voy a detallar más para no ponerme pesado...

En una palabra, tenían una estructura *balcanizada*, con multitud de laboratorios de Divisiones muchas veces enfrentadas entre sí, de gama de productos, muchos de los cuales no eran rentables, pero que se mantenían en catálogo... y además, comercialmente, lo que había sido su lado fuerte de toda la vida, no lo estaban haciendo bien tampoco.

Por ejemplo (y *esto lo he visto yo con mis propios ojos*), tú eras un cliente cualquiera y hacías una petición de ofertas para un nuevo sistema a diversos fabricantes. Recibías una de Hewlett-Packard, otra de Digital, otra de Sun... y **tres** de IBM: una con un buen mainframe, otra con un AS/400 y otra con RS/6000. La fuerza comercial de Hewlett-Packard defendía a muerte su oferta, la de Sun, la suya... y la de IBM defendía *tres*. Contradictorias, incompatibles entre sí. Cuando venía el técnico de preventa de AS/400 a defender la suya, te decía que eso del UNIX no valía para nada, incluyendo, sin citarlo, al RS/6000 con AIX, claro. Después llegaba el del mainframe y te aseguraba que cualquier sistema pequeño no te serviría (ninguno, tampoco los de IBM) y sólo un buen mainframe podría funcionar en tu caso... y así. Comercialmente hablando es una postura suicida... y que a poco le cuesta la desaparición a IBM.

La verdad es que la gestión de John Ackers, Presidente y CEO de IBM durante esos años cruciales, fue bastante nefasta para ellos; no supieron adaptarse a los cambios del mercado, parte de los cuáles ellos mismos habían provocado, no previeron el comportamiento futuro del mercado... o quizá es que *el mercado no se comportó como ellos pensaban*. El caso es que las cifras negativas estaban ahí. Eran persistentes. ¡Y gritaban!

Además, desde mi punto de vista, lo más curioso es que **ellos mismos habían llegado a creerse todo el argumentario de la competencia** y se habían convencido de que la tecnología de mainframes estaba obsoleta y había que desmontar o vender al mejor

postor toda la División, así que estaban preparando (de hecho, estaba el Plan ya muy avanzado) una disgregación de la compañía en los diferentes negocios para venderlos después uno a uno, al mejor estilo de Richard Gere en *Pretty Woman*...

Sin embargo, entre los accionistas de IBM alguien seguía teniendo las ideas claras: en abril de 1993 contrataron a un nuevo CEO, o sea, el que manda más: Lou Gerstner. En el mundillo informático nos causó bastante estupor: este Señor provenía de RJR Nabisco, una compañía alimentaria especializada en vender *galletas y tabaco*... **¡no sabía nada de dirigir una empresa de tecnología!**

El que más, el que menos, pensamos: “*Buff, estos de IBM se han vuelto definitivamente locos: se van a la quiebra en dos días...*”

Pues, mira tú por dónde, no. Resulta que Lou Gerstner sabía sobre todo de dos cosas: cómo **funcionaba el mercado de consumo y cómo organizar una empresa enorme y variopinta**. Y, además, conocía muy bien a IBM como cliente suyo: estaba al cabo de la calle de sus fortalezas y de sus debilidades... Y eso era exactamente lo que entonces necesitaba IBM.

Canceló toda segregación de la compañía, imponiendo a cambio un régimen de adelgazamiento brutal en las áreas que no eran rentables.

Pero sobre todo, *sobre todo*, se dio cuenta de que el **negocio tradicional de IBM había cambiado**, que era lo que no habían sabido ver sus antecesores. IBM había ganado dinero a espuertas durante toda su historia, fabricando unos cacharros excelentes y vendiéndolos con elevados márgenes que pudieran compensar los costes de desarrollo. Pero ese modelo de negocio basado en el hardware estaba tocando a su fin: **el futuro era de los intangibles: el Software y los Servicios Profesionales**. Y lo sigue siendo, por cierto.

Vale que hacen falta máquinas para que ese Software funcione, pero cada vez más son consideradas como puros *commodities*, intercambiables, con escaso valor añadido. En el mundo de los PC's, desde luego, pasa ya, pues lo último que miras al adquirir un PC es su marca, ¿cierto?, y en buena medida pasa también con los servidores UNIX o Windows...

En muy poco tiempo IBM se convirtió, de ser un proveedor virtualmente monomarca (*la suya*, claro), a uno de los integradores de Sistemas más activos, compitiendo con Andersen Consulting, Coopers & Lybrand, Price Waterhouse, etc., en su propio terreno: la **Consultoría e Integración de Sistemas**. Firmó acuerdos con prácticamente todos los fabricantes de cualquier cosa. Y, encima, contraviniendo toda su historia anterior, los puso en acción.

Los clientes, atónitos, comenzamos a recibir ofertas de IBM donde había, por ejemplo, una máquina Sun E10000 con Solaris, Base de Datos Oracle, algún otro producto de gestión de IBM o de terceros y Servicios Profesionales de IBM, lo que luego se convertiría en **IBM Global Services**. No nos lo podíamos creer, ya digo: “*El Acabóse*”...

Pero... **funcionó**.

Pusieron en la calle a decenas de miles de personas con unas condiciones de salida realmente interesantes, casi todas ellas técnicos de sobrada valía que casi en su totalidad encontraron rápidamente trabajo entre sus clientes u otros proveedores. Varios compañeros míos proceden de aquellos años y doy fe de que son unos excelentes profesionales.

Parecía que IBM, sin sus mejores técnicos y comerciales, no duraría mucho.

**Duró.**

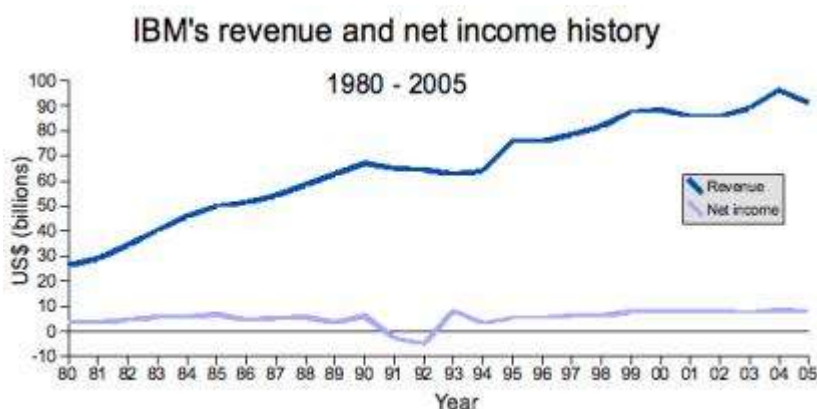
A mediados de los noventa volvió a ganar dinero. Poco, quizá, comparado con las ingentes cantidades de años anteriores, pero eran números negros. Y así, creciendo, siguen hasta hoy. Mi personal opinión es que IBM va a ser con toda probabilidad una de las

ganadoras de la *RecojoCrisis del 2008-09-10-11-12-13-14-xx??*

Aunque ciertamente no estoy yo seguro de que la política llevada a cabo por IBM en los últimos años sea la adecuada: desarrollar muy poco, pero comprar decenas o centenares de pequeñas (o no tan pequeñas) compañías emergentes con productos resultones y una cierta base de clientes instalada... Es una estrategia que me recuerda muchísimo a la de Computer Associates durante tantos años. Y no puede decirse que CA haya sido una de las vencedoras de la revolución tecnológica global... En fin, el tiempo nos lo dirá.

Por de pronto, a continuación tenéis un gráfico con la evolución de Ventas y Beneficios de IBM desde 1980 a 2005. ¿Se nota el bache brutal en ambas cifras de los primeros años noventa, coincidiendo con la *crisis de esos años*, y cómo se recuperó en años sucesivos?

Mérito de Mr. Gerstner.



Mientras todas estas cosas pasaban, algunos clientes habían empezado proyectos serios de downsizing, que siempre tiene que haber alguien que empieza... y, como podéis imaginaros, **las cosas no eran tan sencillas como el corifeo de consultores aseguraba**, ni con todos los técnicos posibles echando una mano. Ni el hardware ni el software estaban maduros, ni tampoco los profesionales tenían todos los conocimientos necesarios para asegurar el éxito de los proyectos.

Los que arrancaron estos proyectos esperaban un desarrollo rápido, de bajo coste y, sobre todo, una aplicación resultante al menos tan eficiente como la sustituida, más barata y sencilla de mantener y, además, con colorines... pues prácticamente todas estaban escritas siguiendo la tecnología **Cliente/Servidor**, que era el otro gran hallazgo *marketiniano* del momento, es decir, los usuarios manejaban un PC con ratón y ventanitas de colores, conectado generalmente mediante una red de área local con un servidor.

Hubo algunos descalabros importantes, yo conozco alguno de ellos; hubo quien tuvo que dar marcha atrás y todos los proyectos, sin excepción, duraron mucho más de lo previsto, costaron sensiblemente más de lo presupuestado y pocas veces cumplieron las expectativas de funcionalidad y coste de mantenimiento.

Pero es que, además, **muchísimos Usuarios Finales habían comenzado a mecanizar sus aplicaciones sin la intervención del Departamento de Informática**. Y es natural: los Usuarios tienden a desear que se solucionen sus problemas de forma rápida, sencilla y barata, y cuando nos explicaban sus necesidades a los de Proceso de Datos, tras estudiar sesudamente el asunto les dábamos plazos de varios años y costes monstruosos para resolverlos... pues no les gustaba. Pero *nada de nada*.

Muchos de ellos (de los Usuarios, quiero decir) buscaron alternativas y se *buscaron*

la vida, dotándose de aplicaciones que resolvían, más o menos, sus problemas inmediatos. Usaron diferentes estrategias, desde contratar directamente a algún suministrador sin el conocimiento de Informática (o *con su conocimiento y subsiguiente enfado*), o bien, y esto era lo más normal, resolviéndoselo él solito, con una buena Hoja de Cálculo toda llena de macros, una Base de Datos Microinformática o directamente encargando la aplicación a su sobrino, estudiante de informática en una academia y *mago* del Basic.

Todas estas aplicaciones “*extraoficiales*” se montaron sin excepción en PC's, casi siempre aislados y en algún caso con servidores, también PC's, rara vez una máquina UNIX. Nunca, nunca en mainframe, ni siquiera en Sistemas UNIX con las Bases de Datos, ya bastante robustas, de la época. Y claro, cuando hubo que ampliar, expandir o modificar estas aplicaciones que, a lo tonto, a lo tonto, se habían convertido en críticas para el negocio... pues hubo problemas. Serios problemas.

Así que, inevitablemente, hubo una reacción “*anti-downsizing*”, y un nuevo palabro apareció en escena: el *upsizing*. Me extraña, por cierto, que lo que decía la Wikipedia sea verdad: que fue Microsoft quien acuñó el término refiriéndose a migrar aplicaciones de Access a SQL Server... a lo mejor sí, pero desde luego aquí en la Piel de Toro no se usaba para eso... al menos, no exclusivamente para eso.

Con ese término nos referíamos al **movimiento de las aplicaciones distribuidas** (muchas de ellas, *desparramadas* más que distribuidas) **a un entorno centralizado**, no necesariamente mainframe, aunque también.

En efecto, esas aplicaciones basadas en PC, baratitas y con no muchas pretensiones en su origen, pero que con el tiempo se habían convertido en aplicaciones importantes para las empresas... fue entonces cuando comenzaron a mostrar crudamente los problemas de esa aproximación *pecera*: falta de integración, diversidad de versiones, incompatibilidad con otras aplicaciones, dificultad de acometer nuevas versiones o añadir funcionalidad nueva, graves problemas de recuperación en caso de fallo... o sea, todo lo que nos había costado muchos años aprender a base de tortas en el *entorno centralizado de toda la vida*, que últimamente se nos había olvidado... y que hubo que recordar a marchas forzadas.

También se hicieron proyectos de upsizing y también con problemas, claro. Porque ahora sí que había integridad de datos y funciones... pero la interfaz no cubría las necesidades de los usuarios, que habían visto, o al menos entrevisto, lo útil que puede llegar a ser un buen sistema de ventanas, con su ratoncito y sus botones y eso.

Y así llegamos, finalmente, a inventar de una vez el *término definitivo*: ¡**Rightsizing!** Es decir, más o menos: “*Seleccione la tecnología y ubique las funciones de su Sistema de tal modo que se usen las diferentes máquinas y software disponibles para lograr un rendimiento, interfaz y coste perfectos*”. Es decir: ¡**Acierite Usted!** Sin bola de cristal, eso sí.

¡*Pues vaya!* Nosotros, los informáticos viejos, que pensábamos que exactamente *eso* es lo que habíamos estado haciendo toda la vida... y resulta que ahora se llamaba “rightsizing”, cuando creíamos que se llamaba *trabajar*...

En fin. Ninguno de estos términos tuvo éxito a la larga, porque, a la corta, ya lo creo que sí. De hecho, ni siquiera se encuentran sus definiciones adaptadas a la tecnología informática en la Wikipedia, es decir, se ve que ahora lo llamamos otra vez *trabajar*... Porque para eso nos han pagado desde que la profesión existe, digo yo.

El caso es que los noventa vieron cambiar por completo el papel de los elementos tecnológicos informáticos: **el hardware dejó de ser importante** y comenzó a convertirse

en lo que hoy es: una commodity, y sin embargo **el software se convirtió en lo realmente importante**, en lo que hacía la distinción entre un sistema bueno y uno malo... Eso y **los servicios profesionales**, desde luego.

E Internet, claro. Con la progresiva introducción y aceptación de Internet y la sensible mejora de las capacidades de redes y líneas telefónicas, el uso de un navegador prometía, de una vez por todas, lo mejor de los dos mundos: Aplicaciones centralizadas, sí, con todas sus ventajas de control, disponibilidad, etc., pero con presentación gráfica ante el usuario, y todo por un coste muy ajustado, sobre todo de licencias, dado que en el equipo cliente no se necesita ningún software adicional al mínimo preciso para que funcione: Sistema Operativo y Navegador. Como consecuencia, todo el mundo comenzó a hablar de Internet y a probar y montar su pequeña página web, escribir sus primeras aplicaciones intranet... pero ésta es otra historia y será contada en otro momento.

Por fin, tal y como se estaban abaratando los PC's así como el software, y tal y como estaban apareciendo más y más programas orientados al usuario doméstico, empezando por los juegos, claro, muchísimos nos empezamos a comprar un PC para nuestra casa, con la excusa de llevar las cuenta domésticas, escribir las cartas a tu primo el de Valladolid (con el que hace seis años que no te carteas y del que ni siquiera sabes si se ha casado o sigue aún soltero) y aprender a programar en no sé qué esotérico lenguaje... aunque en realidad todos lo usamos para jugar al Quake, el único y definitivo **Quake** que funcionaba originalmente en MS/DOS...

Mi primer PC de casa (mediados de 1996) fue un clónico con un Pentium-a-secas de 166Mhz, 16Mb de RAM, 60 Mb de disco duro, una pantalla VGA de 15", disketera, por supuesto, y lector de CD's, con Windows 95, y todo al módico precio de unas 350.000 pesetas de la época (2.100 Euros)... ¡y eso que ya habían bajado mucho de precio!

En las empresas fue definitivamente durante esos años cuando se puso de moda contratar compañías de Servicios Profesionales para hacer parte del trabajo de desarrollo o de cualquier otra función de Proceso de Datos, incluso el desarrollo completo de los proyectos, aligerando la plantilla propia y subcontratando más y más personas de cada vez más y más compañías...

Había que “*convertir el gasto fijo en variable*”, *mantra* que se oía por doquier y que sonaba muy, pero que muy bien a los directores financieros y de Recursos Humanos: “No se cargue Vd. con costosos e inflexibles *recursos propios*, hombre, subcontrate Vd. todo lo que pueda (mejor: ¡**Todo!**, y mejor aún, ¡**A mí!**) y, en caso de necesidad, siempre podrá Vd. eliminar subcontratación y aligerar gastos sin incurrir en despidos... O sea, pague Vd. a precio de Oro el programador a la empresa *TalyCual, SA*, que ya se encargará esa empresa de pagar al susodicho programador un sueldo de hambre... pero, eso sí, cuando Vd. lo desee, *TalyCual, SA* despide por Vd. al pobre programador y Santas Pascuas...”

No sé si os suena de algo... pues toda esta política del contrato basura informático se gestó durante los primeros noventa; hasta entonces había poca subcontratación, en España, me refiero, que es lo que yo conozco. Y la triste realidad es que, para mantener las aplicaciones funcionando, se necesita siempre un plantel determinado e irreducible de técnicos, o sea, que eso de que los gastos eran variables... sólo lo son hasta cierto punto.

Y exactamente eso es lo que están viviendo muchas empresas en nuestros atribulados días: ahora que con la *consabida crisis* ha llegado el momento de eliminar gasto variable... resulta que **no se puede**, al menos no se puede *tanto como querían*, porque **¡alguien tiene que mantener funcionando las aplicaciones de la Compañía!** Al menos,

*mientras exista* la Compañía, que tal y como están las cosas, nadie puede asegurarlo.

Y así andamos...

En los próximos capítulos hablaré del nacimiento de uno de los *conceptos clave* de la informática de nuestros días: **El Data Warehouse** y sus *colaterales*: el Data Mining, el Business Intelligence, etc.

Como voy a entrar en cierta profundidad en estos temas, esta historia me llevará cuatro capítulos completos. Es una historia larga y compleja, y la tecnología involucrada es realmente muy importante en el mundo de hoy, así que bien lo merece. Creo yo.

## 17 - Cuando descubrimos el “procesamiento paralelo”

Los capítulos anteriores estuvieron dedicados a las convulsiones de todo tipo que acontecieron en los tormentosos años noventa, que acabaron con el cambio del foco de la rentabilidad, que pasó del hardware (que tradicionalmente había sido el máximo generador de beneficios) al software y los Servicios Profesionales, donde hoy sigue.

Pero lo más importante quizá de esa década tan movida fue que se vivió un enorme crecimiento de las aplicaciones informáticas instaladas en todas las empresas. Si a principios de la década podían existir empresas (no muy grandes, desde luego) sin apenas ninguna aplicación informatizada, a finales era ya imposible: o tenías tus aplicaciones principales soportadas en sistemas informáticos, o estabas *muerto*... o quebrado.

La cantidad de información de que se disponía almacenada en los ordenadores de las empresas e instituciones era cada vez mayor, y creciendo de forma exponencial. Centenares de millones de registros, miles de millones... Y claro, los usuarios querían acceder a toda esa información para aprender cada vez más sobre sus clientes, sus productos, sus proveedores, sus empleados... su negocio, en definitiva. Parte de esta información era fácilmente accesible, pero otra, no tanto... y otra más, la gran mayoría, era totalmente inaccesible: estaba enterrada entre decenas de bases de datos, con estructuras en buena medida incompatibles entre sí... así que la información, *estar*, lo que se dice *estar*, estaba... pero **estar**, lo que se dice **estar**... pues no estaba. Se me entiende, ¿no?

De las causas de esta situación, de cómo la tecnología comenzó a encontrar soluciones, o mejor, a aplicar soluciones que ya había para otras cosas a estos problemas, y de cómo empezamos a resolverlo tratan este capítulo y los tres siguientes.

Efectivamente, a principios de los años noventa había ya en producción muchísimas de las aplicaciones básicas de las organizaciones de todo tipo. Y durante esa década, mientras pasaban todas esas cosas que os contaba en capítulos anteriores, se estaban escribiendo muchas aplicaciones nuevas y también rescribiendo las que se habían quedado obsoletas, bien con las mismas tecnologías que sus predecesoras, es decir, casi siempre en mainframe (Cobol, CICS o IMS y DB2), o bien en otro tipo de servidores, casi siempre UNIX y con la tecnología Cliente/Servidor de la época.

Pero todas ellas, sin excepción, estaban orientadas a la transacción. Es decir, su principal objetivo era, y *sigue siendo*, en realidad, atender a las operaciones de los usuarios con precisión, rapidez y fiabilidad. En una palabra, debían ser aplicaciones eficientes. Y ser eficientes quiere decir que deben ser capaces de tratar las transacciones de muchos usuarios simultáneos y darles el mejor servicio factible con el menor tiempo de respuesta posible.

Por ejemplo, muchos vendedores, cajeros y administrativos de un banco cualquiera, incluso clientes en cajeros automáticos, están haciendo simultáneamente operaciones: abonos o adeudos en cuenta, consultas de movimientos, pago de recibos, altas de contratos, etc. Son todas ellas operaciones relativamente sencillas desde el punto de vista de que cada una de ellas, individualmente, necesita pocos datos (mejor dicho, pocos datos *de cada base de datos*) para cumplir su objetivo, aunque el proceso interno que deban realizar, por ejemplo, para autorizar un pago de cheque, no sea nada sencillo de programar, debido al tratamiento que hay que hacer de todas las posibles alternativas que se puede encontrar la operación.



Si bien esa operación requiere acceder a bastantes tablas diferentes, no tiene que acceder a muchas filas de cada tabla; hay un cierto consumo de CPU por transacción, pero con pocos datos involucrados. Son, pues, aplicaciones donde **cada transacción accede a pocos datos con un consumo moderado-bajo de CPU**, donde el criterio de diseño más importante es **la capacidad de atender simultáneamente a muchas transacciones**.

Las máquinas en que se desarrollaron estas aplicaciones, las de la época, los ochenta y noventa del siglo pasado, no tenían mucha capacidad comparada con la que hoy en día tenemos a nuestra disposición; de hecho era algunos órdenes de magnitud menor. Así que, para conseguir esa eficiencia en las aplicaciones, éstas debían estar diseñadas muy cuidadosamente para agilizar este acceso transaccional.

El diseño de las Bases de Datos, el lenguaje de programación elegido, los gestores de teleproceso, toda la arquitectura hardware y software, en definitiva, estaba orientada siempre a tal fin: que las transacciones fueran rápidas y el rendimiento fuera siempre bueno.

De esta forma, la parte *importante* de las aplicaciones eran siempre los procesos de actualización, las entradas de datos, los cálculos... y luego estaban los flecos, las tonterías, lo accesorio, lo que siempre se deja para el final: **las consultas y los listados**.

Todo programador que empezaba su carrera se tiraba unos meses escribiendo sólo programas de listados y consultas, hasta que se curtía lo suficiente como para dejarle programar las cosas realmente importantes... y esto lo sé por experiencia: *yo fui uno de ellos*.

Naturalmente, todas las aplicaciones permitían a los usuarios obtener datos contenidos en las Bases de Datos mediante consultas programadas como unas transacciones más... bueno, no, como unas *transacciones de segunda categoría*, más bien, que mostraban a los usuarios una cierta información predeterminada negociada con ellos al analizar la Aplicación, una información generalmente de detalle, como los movimientos de una cuenta, o las líneas de un pedido, etc. También se emitían ciertos listados durante los procesos batch con resúmenes de la información, datos contables, relaciones de excepciones, etc.

Estos listados se consensuaban con los usuarios, al igual que el resto de las transacciones, durante el Análisis.

Siempre se pretendía que los listados fueran los menos posibles, para minimizar el consumo de papel... porque se imprimían físicamente en papel continuo, que primero había que cortar y separar utilizando las máquinas específicas para ello que todavía hoy siguen siendo usadas, y después enviar físicamente a sus destinatarios, incluidas las oficinas, y todo ello con un coste tremendo.

Ya sabéis que el *nirvana* del informático es no tocar nunca jamás una aplicación, aplicando el muy conocido aunque nunca publicado *Teorema Fundamental de la Informática*: “**Si funciona, ¡NO LO TOQUES!**”.

En consecuencia con este principio fundamental, procurábamos por todos los medios a nuestro alcance que estos listados nunca más se modificaran, al igual que el resto de la aplicación.



Todo esto estaba muy bien al comienzo de la historia, porque las primeras que se mecanizaron fueron las aplicaciones contables más duras, es decir, aquellas que consumían mayor tiempo y más cantidad de recursos haciendo labores en gran parte mecánicas: de ahí lo de *mecanizar* un Departamento que decíamos mucho en la época.

Aplicaciones como las Cuentas Corrientes, la Contabilidad, la Facturación, la Nómina, los Pedidos, etc., fueron las primeras que se atacaron. Todas ellas tienen una gran cantidad de operaciones al día, pero con una lógica sencilla y fácilmente reproducible por un programa aunque, bueno, viendo las especificaciones funcionales hay veces que es difícil crearlo. Todas ellas, sin excepción, consiguieron importantes incrementos de eficacia en las empresas casi desde el día siguiente al de su implantación.

Recuerdo que, en el primer banco en el que trabajé a mediados y finales de los setenta, había una sala enorme de administrativos que habían sido seleccionados entre todos los empleados del Banco por su buena letra (sí, sí, es cierto, por *su buena letra*, seguid leyendo), escribiendo a mano los títulos del propio banco: sus acciones. Cada vez que alguien compraba acciones de aquel banco, alguien escribía con perfecta redondilla que “D. Fulano de Tal posee 27 acciones del banco Tal y Cual, adquiridas el día tal del año tal, que le han costado tantas y cuántas pesetas, firmado y rubricado, etc.”. No creo que tenga que decir que lo del “Mercado Continuo” no existía, ni los “Apuntes en Cuenta”, ni nada de nada. En una palabra, si tenías tal documento en tu poder tenías las acciones, y si no... pues no las tenías. Por eso era tan importante que el documento estuviera escrito con buena caligrafía.

Entonces fue cuando escribimos la nueva Aplicación de Accionistas, bastante sencillita en cuanto a su lógica, por cierto, que mantenía el fichero de Accionistas del Banco. Con los movimientos del día se daban las altas y bajas pertinentes y se imprimían los títulos escritos por la impresora del ordenador, con letra sensiblemente más fea que la esmerada caligrafía del escribano, pero perfectamente legible.

No sólo se ganó muchísimo tiempo en el proceso, pues ahora todas las operaciones del día se imprimían durante la noche y se enviaban a los accionistas al día siguiente, sino que, de la noche a la mañana, los setenta u ochenta *amanuenses* seleccionados por sus dotes caligráficas pudieron ser asignados a otros Departamentos u Oficinas... no despedidos, desde luego, porque en esos años era muy complicado despedir a nadie y porque además los bancos estaban llevando a cabo un importante plan de expansión y apertura de oficinas.

A eso me refiero cuando hablo de *mecanizar un Sistema*.

Pues bien, durante los años setenta, ochenta y primeros noventa se habían mecanizado todas las *aplicaciones mecánicas* susceptibles de ser mecanizadas, y las pocas que no lo estaban aún estaban en el proceso de serlo y se estaban escribiendo ya algunas aplicaciones que llevaban bastante más “*inteligencia*” incorporada en ellas.

Un buen ejemplo de ellas es el del acceso al Mercado Continuo Bursátil: ahora no se trata sólo de realizar y apuntar mecánicamente las transacciones, sino que los programas tienen que tomar ciertas decisiones (*muchas* decisiones, en realidad), en función de las condiciones concretas del mercado en cada momento, del número de peticiones existentes, etc. Hay muchos, como sabéis, que deciden por sí mismos, sin encomendarse a Dios ni al diablo, si hay que comprar o vender tal o cuál acción o bono nacional... Se trata de programas muy complejos, tanto los que hay en la propia Bolsa como los de los Intermediarios Financieros que operan en ella.

Lo mismo pasa con el Departamento de Riesgos... con el incremento tremendo de tarjetas de crédito en manos de clientes esos años y de las operaciones realizadas diariamente, también se incrementó el fraude, así que hubo que escribir programas que autorizaran las operaciones sin intervención humana... y esos programas tenían también que tomar decisiones, **decisiones que, si son erróneas, cuestan dinero constante y sonante**.

Este tipo de aplicaciones tienen unos requerimientos de obtención de información por parte de los usuarios que van bastante más allá de simplemente consultar cuántas operaciones se han hecho, cuántas se han denegado y cuántas se han concedido y por cuánto importe... Ahora no sólo hay que saber **qué** ha pasado, sino que es imprescindible saber **por qué** ha pasado lo que ha pasado, seguramente porque habrá que intentar conocer de antemano **lo que va a pasar**. Este sutil cambio tiene una consecuencia inmediata: **es virtualmente imposible anticiparse a las necesidades de información que tendrán los usuarios de este tipo de información en el futuro**.

Pero es que ¡eso es exactamente lo que tradicionalmente pedíamos a nuestros usuarios!: que se mojaran, que nos dijeran en el Análisis Funcional, seis meses, o un año o dos antes de que la Aplicación estuviera por fin disponible para su uso, qué listados y consultas debía obtener dicha aplicación, y además le pedíamos que firmara las especificaciones con sangre, porque cualquier cambio posterior le llevaría de cabeza a los infiernos... aunque, en realidad, *exactamente eso es lo que seguimos haciendo veinte años más tarde*, pero en fin.

Y claro, la pura y simple realidad es que ningún usuario de estos sabe qué va a necesitar dentro de seis meses para averiguar por qué una Oficina vende más créditos que otra o por qué tienen más éxito las cacerolas que los juegos de toallas... de hecho, tampoco sabe qué es lo que va a necesitar *mañana*, así que piden lo primero que se les pasa por la cabeza o lo que les preocupa en ese momento concreto, lo firman... y luego, a la postre, no les sirve de nada.

En realidad, a los informáticos se nos entrena desde el principio para este proceder ante nuestro usuario, que podemos resumir en la máxima: “**Habla ahora o calla para siempre**”, sólo que *eso siempre lo decimos muchos meses antes de entregar la aplicación funcionando*.

Todas las Metodologías de Desarrollo habidas y por haber, Orientadas a Objetos, al Norte, al Sur o a donde sea, hacen hincapié, pero que mucho hincapié, en esta cuestión: *las Especificaciones deben estar congeladas antes de comenzar a escribir el software*. Como

decía un insigne consultor que yo conozco, “*Escribir software es lo mismo que andar sobre el agua: es mucho más sencillo si está congelada*”.

*Os voy a contar un secreto:* hay muchas ocasiones en que es imposible que un usuario te pueda decir tales cosas, ni siquiera una semana antes de poner la aplicación en marcha. Y es más, cada semana necesitará nueva información que no se le había ocurrido, o que no había necesitado con anterioridad. Cuando nos toca en suerte un usuario de este jaez, los informáticos tendemos a, primero, enfadarnos con él, y luego catalogarle con frases como “*Este Usuario es un cretino; no sabe lo que quiere*” y empezamos a ningunearle, ponerle excusas, marearle... con tal de que nos deje en paz. Pues, siguiendo con ese secreto a voces que os estaba desvelando, no es que ese usuario no sepa lo que quiere... **¡es que no puede saberlo de antemano!** Es así de sencillo; duro para nosotros, los informáticos, pero muy sencillo.

Mi primer encontronazo con esta cruda realidad, aunque he de reconocer que *entonces* yo también llegué a la sesuda conclusión de que *el usuario no sabía lo que quería*, sólo muchos años después reconocí los hechos, ocurrió a fines de los setenta, en ese primer banco en el que velé mis armas informáticas.

Mi jefe me pidió que atendiera al Responsable de Marketing del Banco, que necesitaba no sé qué listado. Entonces toda la información que se pedía al ordenador había que emitirla en papel, pues ésa era la única manera de sacar de las tripas del Sistema información legible por los humanos. He de decir que el Departamento de Marketing en ese Banco en 1979 era pequeño. *Muy* pequeño. En realidad atendí al Responsable del Departamento, que se componía de una única persona: él mismo. Por cierto, en todos los bancos de la época los Departamentos de Marketing tenían más o menos los mismos recursos... no como unos años después o como ahora mismo, sin ir más lejos.

Bueno, pues lo que quería este buen hombre era un listado de clientes que cumplieran ciertas condiciones para hacerles una promoción, ofertarles algo o no sé qué... tampoco me enteré muy bien para qué lo quería, pues por entonces trabajaba en el banco, pero no sabía nada de banca... después aprendí. Tomé cuidadosa nota de las condiciones y me puse a escribir los programas para extraer la información y listarla, según el procedimiento que os conté en el capítulo correspondiente al método de trabajo en los setenta.

Como los ficheros estaban en cinta magnética no era nada sencillo recuperar esa información: un programa de extracción, un buen Sort, un fantástico Padre-Hijo (otra vez sale a relucir el *Padre-Hijo*...) con el fichero maestro y, por fin, otro programa más de listado. Uno o dos meses después, según cómo de cargados estuviéramos todos, los programas estaban listos, funcionaban y se ejecutaron. El listado resultante tenía *metro* y *pico* de altura. De veras. Lo subimos de la Sala del ordenador en un carrito y casi hunde la mesa en que lo pusimos.



Y estaba *niquelado*, de acuerdo a los requisitos solicitados y debidamente firmados por el dichoso Usuario de Marketing. Le avisamos: “Buenas. Tu listado está listo, ¿qué hacemos con él?”

¡Uy, qué bien! Pero espera, que voy a comprobar cómo está la información...” Va a nuestra oficina y, según ve el volumen del listado y antes siquiera de hojearlo, dice:

*“Ah, pero no me vale, porque necesitaba como máximo diez o quince mil clientes y ahí debe haber muchos más...”*

“Ehh... Sí, exactamente 368.476 clientes, que para eso el programa los ha contado minuciosamente...”

*“Buff, son muchísimos. No, así no me sirve... A ver, podríamos eliminar a los que cumplan esta y esta otra condición, pero añadir además a los nuevos clientes que tengan saldo mayor que tal y cual, y bla, bla, bla”.*

Respuesta evidente e inmediata que cualquier informático hubiera hecho: “Pero... vamos a ver... ¿por qué no me lo has dicho antes?”

*“Eeer, pues... es que me figuraba que habría muchos menos clientes que cumplieran esas condiciones...”*

“Ah, pero... ¡¿no lo sabías?!!! #\$&@#!”

“Nooo...”.

Modifiqué los programas bastante *mosqueado*, puse un contador para contar de antemano, antes de listar nada, cuántos clientes habría que cumplieran todas las condiciones pedidas, aislé en un programa las condiciones aplicadas para que fuera fácil de modificar... un mes más tarde y siete llamadas telefónicas después para afinar la extracción y asegurar que esta vez sí valía, el nuevo listado estaba impoluto, y esta vez de tan sólo quince centímetros de alto, encima de mi mesa.

Nueva llamada y nueva conversación. Ahora no servía tampoco, porque habría que incluir a los clientes que, teniendo Depósito de Valores, no tuvieran Imposiciones a Plazo, pero sí más de una cuenta corriente y bla, bla, bla... Grrrrr!!!!

No recuerdo si llegué a modificar de nuevo los programas, o si se tuvo que apañar con el listado que tenía, o si simplemente no lo usó para nada al final... Nosotros le catalogamos como *persona non grata* en Proceso de Datos, le relegamos al olvido y él no volvió nunca jamás a pedirnos nada, al menos mientras estuve yo trabajando allí.

Muchos años y varias empresas después, **comprendí por fin lo que ocurría**: por un lado, su falta absoluta de información sobre datos tan básicos como cuántos clientes cumplen ciertas condiciones sencillas (el pobre hombre realmente no sabía si había 10.000 ó 300.000 clientes que las cumplieran, lo que supone un rango de error realmente considerable), y, por otro, las cambiantes condiciones de un mercado cada vez más competitivo hacen que la información que hoy es absolutamente prioritaria... *mañana sencillamente no vale nada*.

Nuestros estupendos programas eran capaces de abonar un dividendo con perfección matemática, calcular los intereses de forma exacta y renovar los depósitos con precisión milimétrica... pero en realidad **no sabían nada sobre el negocio**, nada de nada sobre lo que de verdad mueve el negocio de la empresa. Así eran las cosas, amigos, y así siguen siendo en muchísimos casos.

Pues bien, después de todo este *speech*, resulta que **los Usuarios de verdad**, los que toman las Decisiones Realmente Importantes para la empresa, **necesitaban una información de gestión de mucha mayor calidad** para la toma de decisiones de negocio **que la puramente transaccional**.

Porque lo que ocurre generalmente es que las *Decisiones Realmente Importantes* se tomaban como lo hace el pobre usuario de la ilustración...



Y a finales de los años ochenta era ya evidente que así no se podía seguir.

Habíamos *mecanizado* todas las aplicaciones susceptibles de ser mecanizadas, sí, pero teníamos grandes carencias en aquellas que requieren elaborar la información de manera diferente, acumular, extrapolar, obtener estadísticas que permitan a los usuarios *oler* las tendencias del negocio para anticiparse a posibles problemas... o a la competencia. Por ejemplo, los responsables de Riesgos necesitan continuamente conocer con detalle qué está ocurriendo, sí, pero sobre todo **por qué ocurren las cosas**, para adaptar los criterios de concesión o denegación de crédito a las cambiantes condiciones del mercado o, también, a la habilidad de los defraudadores para encontrar los puntos débiles del Sistema para sacar partido de ellos.

El resumen de todo esto: **Las aplicaciones informáticas tradicionales no servían para estos menesteres** y, aún peor, **el propio método de creación de aplicaciones informáticas no servía para resolver estos problemas**.

Desde temprano los fabricantes y consultores intentaron dar solución a esta problemática: a fines de los ochenta se puso de moda el concepto de **Centro de Información**. La idea era dotar de así llamadas “Herramientas de Usuario Final” a los usuarios con tales necesidades y permitirles el acceso a las Bases de Datos corporativas a su albedrío.

Esas herramientas eran normalmente productos microinformáticos con acceso a la información contenida en los ordenadores centrales, aunque había también alguna que funcionaba directamente en mainframe, como el QMF, de IBM y herramientas similares de otros competidores, como SAS.

### **Fue una mala idea.**

Primero, porque si las Bases de Datos no eran Relacionales, y esos años había muchísimas todavía, la cosa se ponía realmente fea; pero aunque se pudiera acceder, las herramientas usadas (Bases de Datos Microinformáticas, como Dbase II o III, por ejemplo), no eran ni muy potentes ni muy eficaces accediendo a la información contenida en, por ejemplo, un mainframe. Pero además es que había un problema de base, puramente tecnológico, que desaconsejaba acceder directamente a las Bases de Datos de Producción: tal acceso no sólo tenía un rendimiento penoso, sino que además *destrozaba* literalmente el rendimiento de las aplicaciones transaccionales, que, cuidadas y mimadas como estaban, tan bien iban y tan buenos tiempos de respuesta proporcionaban... con las máquinas de la época.

Así que IBM comenzó a promocionar, dentro de su concepto de “Centro de Información”, que **los clientes se compraran otro mainframe** y diariamente (o semanalmente, o con la periodicidad que fuera) **duplicaran sus bases de datos corporativas** para que luego los usuarios finales pudieran acceder a ellas y manipularlas a su antojo sin comprometer el rendimiento del sistema transaccional. Una política muy interesante... *para IBM*, claro.

Incluso vendieron alguno; yo mismo, de hecho, estuve en Alemania viendo cómo BASF había duplicado su parque de mainframes para dar acceso a sus usuarios finales con su QMF... En España *no coló*. Ninguno vimos claro que semejante inversión sirviera para algo.

No conozco ninguna empresa que comprara nuevos mainframes exclusivamente para esto, aunque igual hubo alguna. Siempre hay alguna para cada algo... Y, total, un mainframe es un mainframe y siempre se podía reaprovechar para absorber el crecimiento vegetativo.

En cualquier caso, quizá os preguntéis que por qué no es bueno que a un mainframe (u otra máquina cualquiera) le venga tan mal que, mientras hace su trabajo normal, sus transacciones y su batch, alguien esté accediendo a las Bases de Datos con las así llamadas *queries libres*... Como la razón última de todo esto tiene muchísimo que ver con la propia filosofía de diseño de los sistemas, creo que es importante explicarlo bien.

Pero para ello me temo que *no me queda más remedio que daros una conferencia*... pero que será útil: en ningún sitio podréis encontrar lo que viene a continuación: es *invento* de este humilde informático vejstorio y pelmazo. Quizás sea una tontería, sí, pero es *mi* tontería... y a mí me parece que es importante fijar bien estos conceptos para que comprendáis todo lo que sigue. Empezamos...

Bien, un Sistema Informático cualquiera, entendido como el conjunto de un determinado hardware y un cierto software configurado de alguna manera para sacar el máximo partido al hardware, tiene una cierta **potencia agregada**, definida como bien os plazca. Y, por otra parte, cualquier Sistema Informático debe lidiar, simultáneamente, con tres tipos de **cargas de trabajo**:

- 1) **El número simultáneo de usuarios a los que da servicio.** Un ordenador

monotarea sólo hace una cosa a la vez; mientras que un sistema diseñado para ser online deberá ser capaz de dar servicio a muchos usuarios o trabajos batch simultáneos.

2) **La complejidad de cada petición individual.** Es decir, la potencia de cálculo requerida para dar servicio a cada transacción, programa batch, etc.; a cada unidad de trabajo que se le solicite, en definitiva.

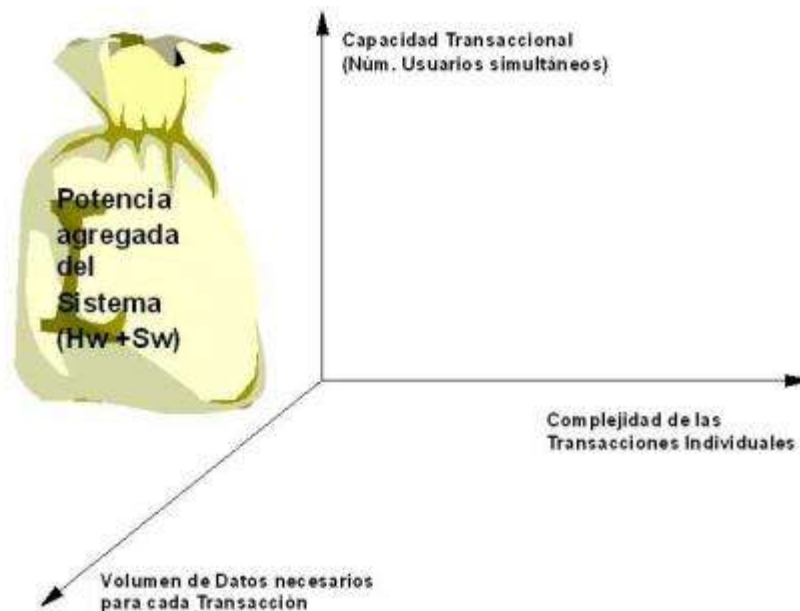
3) **La cantidad de datos a los que acceder por cada petición individual.** Una transacción online no debería necesitar acceder a muchos datos, mientras que un trabajo batch con cierta envergadura suele necesitar acceder a muchísimos de ellos.

Si ahora representamos estas tres características en un sistema cartesiano de coordenadas obtendremos un espacio tridimensional como el que aparece en las imágenes siguientes, con cada uno de los tipos de cargas de trabajo en un eje, y es en ese espacio tridimensional donde debemos acoplar la **potencia agregada del Sistema** inherente a sus capacidades físicas. Podemos imaginarnos esa “Potencia agregada” como un *saco de arena* que debemos distribuir en nuestro *rinconcito* particular definido por esos tres ejes. Cuanta mayor es la capacidad de nuestro sistema, más arena habrá en el saco, más metros cúbicos de arena tendremos disponibles para jugar con ellos.

La arena contenida en el saco representa, de alguna forma, toda la potencia *nominal* de nuestro sistema. Ahora, según la organicemos a lo largo de cada eje, así funcionará nuestro sistema final. Esto es lo que normalmente se llama *configurar el sistema*.

Esta potencia agregada nominal está, en la imagen, contenida en el saquito de arena...

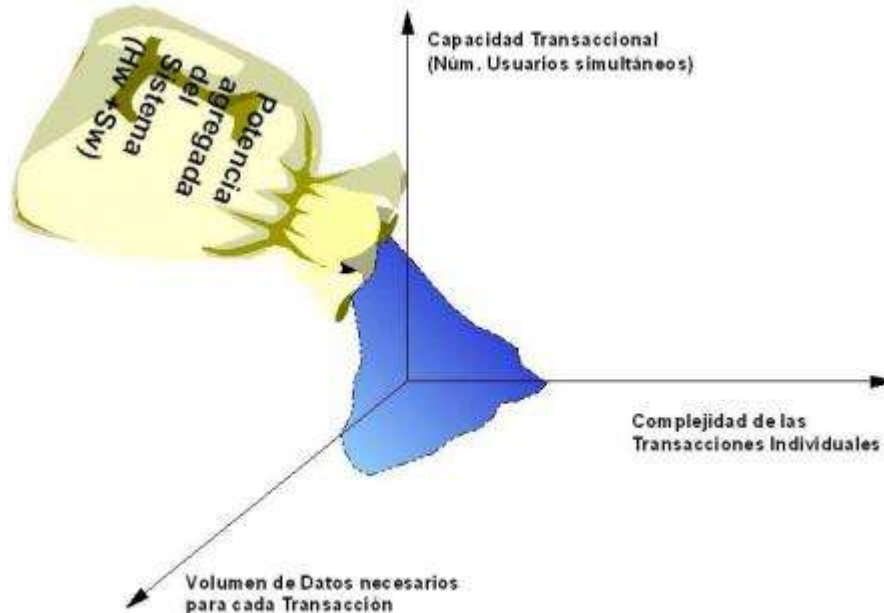
Ingenuo, ¿verdad? Sin duda, sin duda... pero efectivo, muy efectivo, ya lo veréis...



Pues bien, la configuración del sistema (sobre todo la de su software, aunque también la de su hardware) permitirá diferentes soluciones, dependiendo del tipo de trabajo al que queremos dedicar nuestro sistema. Es decir, **cuando se configura la máquina estamos**, de alguna manera, **decidiendo cómo repartir la potencia intrínseca del sistema entre los tres ejes**, o sea, cómo distribuir los *granos de arena* a lo largo de los tres ejes, las tres aristas de nuestro *rinconcito*.



Así que volcamos nuestro saquito y comenzamos a repartir nuestra arena por el dichoso rinconcito como se ve en la imagen:



Veamos: un **Sistema de Propósito General** (un mainframe típico es el ejemplo más evidente) debe ser capaz de resolver bien un *ambiente de cargas mixtas*: por una parte deberá ser capaz de dar servicio a un sistema transaccional con muchos usuarios simultáneos online, pero en el mismo momento también deberá estar ejecutando un cierto conjunto de trabajos batch, todo ello con una cierta complejidad de cálculo.

Es decir, en el caso de un Sistema de Propósito General como el definido lo que nos interesa seguramente es que **nuestra potencia esté lo más regularmente distribuida posible a lo largo de los tres ejes**.

En la siguiente imagen podemos apreciar cómo quedará repartida la potencia agregada de este tipo de Sistema en el rinconcito.



El pequeño cubo (*hexaedro*) del centro, con su vértice, representado en la imagen por el punto más grueso ubicado en la superficie de la figura, representa una posible distribución más o menos regular: un cierto número de usuarios simultáneos, una determinada complejidad de cada transacción individual y un cierto volumen de datos accedido por cada una.

Evidentemente, si queremos dar servicio a más usuarios simultáneos en este Sistema no hay más remedio que reducir la complejidad de cada transacción individual y/o la cantidad de datos necesarios para cumplimentar cada transacción; si deseamos, en cambio, ejecutar un proceso que requiera un considerable volumen de datos para completarse habrá que reducir el número de usuarios simultáneos y/o la complejidad, etc.

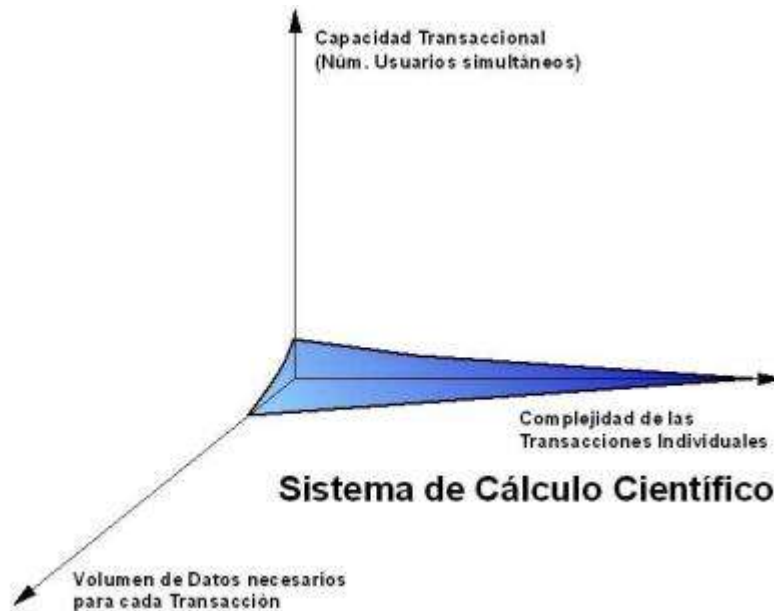
Y todo, claro está, hasta un cierto límite: los puntos de los ejes hasta los que alcanza la figura triangular que representa los *granos de arena* de la potencia agregada: no es posible llegar más allá, el Sistema no tiene potencia suficiente.

Si necesitamos llegar más allá del límite (más usuarios, más complejidad o más datos a acceder), hay que tomar una medida radical: El sistema con esta configuración de “Propósito General” no sirve y tenemos que conseguir otro que pueda llegar más allá. Una solución obvia es incrementar la potencia del sistema en todos sus ejes de forma regular, o sea, *echar, regularmente, más arena al rinconcito*. Ésta es una solución muy utilizada en nuestros días: se pone más memoria, más disco y más rápido, se pone un procesador más potente, o unos pocos procesadores más... y ¡hala! El abaratamiento del hardware nos ha hecho vagos, más de lo mucho que ya éramos antes. Pero cuando el hardware era caro había que *estrujarse las meninges* para obtener más rendimiento para problemas concretos, sin necesidad de adquirir más y más hardware... lo que además, en ocasiones, directamente no era posible.

Efectivamente, existe otra posible solución, también evidente aunque mucho más difícil de conseguir: distribuir la *arena de nuestro saquito* de otra forma a lo largo del *rinconcito* echando más arena en uno de los ejes, a costa de dejar los otros dos con mucha menos. Es decir: **reconfigurar el Sistema para potenciar uno de los ejes** a cambio, claro, de reducir la capacidad del sistema en los otros dos.

Un ejemplo evidente de esto son los **Sistemas Científicos**, diseñados para resolver complejísimos problemas científicos como son la predicción del clima, la investigación genética, la simulación nuclear, etc. Requieren una potencia de cálculo enorme para cada una de sus tareas... aunque con una relativamente escasa necesidad de acceso a datos y donde apenas hay usuarios simultáneos; es más, es muy normal que haya solamente uno: muchos de estos sistemas de miles de procesadores son dedicados y, por tanto, monousuario.

En una palabra, un sistema de tal tipo necesita que nuestro proverbial saco de arena se distribuya a lo largo del eje de la potencia de cálculo como podemos observar en la imagen siguiente.



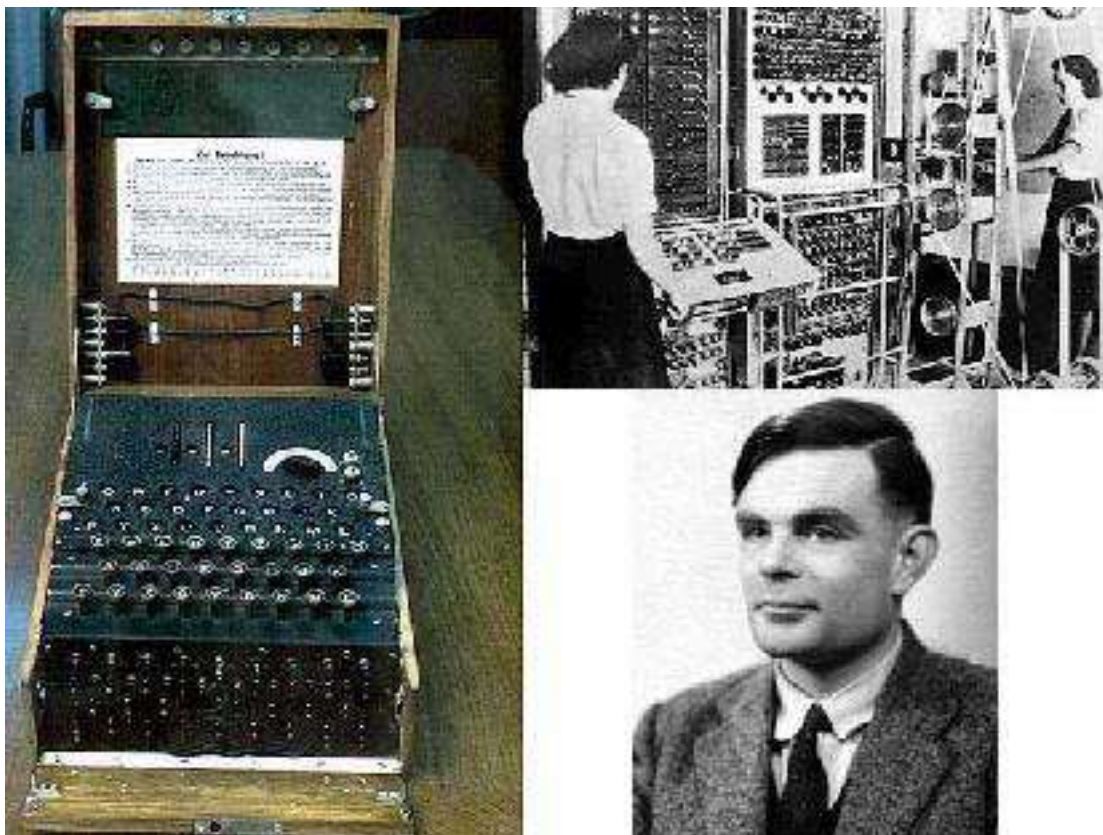
Como sabéis, en realidad buena parte de los avances en la creación de los ordenadores fueron debidos a la necesidad de resolver problemas de este tipo.

Entre ellos está el considerado quizá como primer ordenador moderno, *Colossus*, en Bletchley Park, Reino Unido, diseñado por un grupo de científicos entre los que se encontraba uno de los padres de la moderna informática: Alan Turing. Colossus fue diseñado *ad hoc* para romper el código de las máquinas criptográficas usadas por el Ejército y la Marina Alemanes: **Enigma**.

Este es un trabajo que ejemplariza la definición que antes hice: trabaja sobre muy escasos datos, que se leen al principio del trabajo que es, eso sí, intensivo en cálculos, y todo ello en un sistema especializado y dedicado.

Lástima que Colossus fuera no sólo clasificado como Alto Secreto, sino totalmente destruido al acabar la guerra... ¿a qué paranoico se le ocurriría tal cosa?, y eso que los aliados habían ganado la Guerra...

En el *collage* de la imagen siguiente, a la izquierda, la razón de tantos dolores de cabeza: la famosa **Máquina Enigma** y, a la derecha, **Colossus** en plena operación y **Alan Turing**, debajo.



Durante los años de la Guerra Fría, sobre todo desde 1950 en adelante, los esfuerzos militares para mejorar el armamento o la inteligencia fueron muy intensos y en muchos casos requirieron la creación de nuevas herramientas informáticas para realizar los complejos cálculos requeridos. Por esta razón estos superordenadores recibieron mucha atención (es decir, fondos) por parte de los Gobiernos, sobre todo del estadounidense.

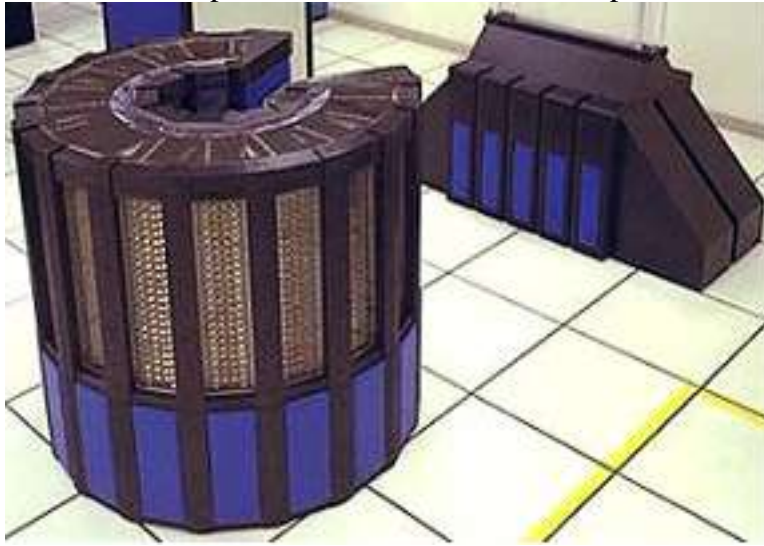
En la década de los sesenta, la compañía norteamericana Control Data Corporation comenzó a construir ordenadores vectoriales, lanzando en 1964 el primer superordenador que es merecedor de tal nombre: el **CDC 6600**. El ingeniero que diseñó esa maravilla, que era nada menos que diez veces más rápido que cualquier otro ordenador existente en la época, fue Seymour Cray, que en 1972 abandonó CDC, a la sazón con serios problemas financieros, y fundó **Cray Research**, la referencia hasta finales del siglo pasado en lo que se refiere a supercomputadores vectoriales. El primero fue el famoso **Cray 1**.

Cray-1 fue el primer ordenador que realmente explotó el *proceso vectorial*, obteniendo velocidades de proceso espectaculares para la época.

Su diseño estaba tan cuidado que incluso fue construido en forma cilíndrica, cosa muy poco habitual en los ordenadores de entonces y de ahora, con el objetivo de acortar los cables que conectaban las diversas partes del sistema (CPU, memoria, registros, etc). No es que intentaran ahorrar costes poniendo menos cables (que, a lo mejor, también, porque llevaba kilómetros y kilómetros de cable...), es que así acortaban el espacio que debía recorrer la electricidad cuando circula a la velocidad de la electricidad entre componentes, reduciendo la latencia del sistema... y además, de paso *resulta un diseño realmente bonito*, para qué negarlo.

Años más tarde, a mediados de los ochenta, lanzó el **CRAY 2**, que era capaz de

alcanzar una asombrosa potencia de cálculo de 1,9 Gflops.



Este sistema Cray 2 fue vendido masivamente a las Agencias Estadounidenses, a todas ellas, incluidas por supuesto las más secretas de todas (éstas fueron, además, sus mejores clientes, por más que Cray no pudiera publicitarlo, dado que se trataba de información estrictamente “*Classified*”), la NASA (el de la imagen es precisamente de la NASA), Laboratorios, Universidades, etc., y también vendió bastantes máquinas en Europa. En España, que yo sepa, no: ni sabríamos muy bien qué hacer con ella, ni creo que pudiéramos pagarla.

Tras el éxito de Cray, otras empresas se dedicaron a fabricar y vender ordenadores vectoriales de computación en paralelo, como Convex o Thinking Machines en EEUU, o NEC, Hitachi y Fujitsu, en Japón.

Hubo un gran *ruido vectorial* en el mundo... pero **no había mercado para tanto supercomputador** y, sobre todo, **la propia evolución de los sistemas normales fue comiendo el terreno a estos monstruos**... que cuando hoy los miramos nos parecen de juguete, pues cualquier ordenador *normalito* de hoy en día, incluyendo el que tú, querido lector, tienes en tu casa o en tu oficina, tiene seguramente mayor capacidad de proceso que muchos de estos superordenadores primigenios.

Eran máquinas especializadas (*especializadísimas*, diría yo), carísimas y necesitaban de una cuidadosa programación, generalmente en Fortran, que tenía una extensión llamada “Fortran Paralelo” para optimizar el uso de las capacidades vectoriales de los sistemas.

Además, no todos los problemas se prestaban para su resolución en este tipo de Sistemas. Más bien al revés, eran pocos los problemas que se prestaban para sacar auténtico partido a estos superordenadores. Y desde luego, entre ellos no se encontraban los Sistemas de Proceso Online de la Información, los **Sistemas Transaccionales de Alto Rendimiento**.

Que yo sepa, los primeros Sistemas Online de Alto Rendimiento que se pusieron en marcha fueron los de reservas de las Líneas Aéreas, en concreto el sistema **Sabre**, realizado por IBM para American Airlines a principios de los 60 del siglo pasado, que en 1964 era ya plenamente funcional. La cantidad de reservas que era capaz de atender por segundo era ridícula comparada con las que hoy en día consigue cualquier Sistema, pero en la época y con las máquinas de entonces y las líneas telefónicas de entonces era un reto de muy difícil



solución.

En el diseño de este Sistema se tomaron medidas radicales como, por ejemplo, no mantener colas de transacciones de entrada. Las transacciones provenientes de los operadores de reservas, cuando llegaban al sistema, si podían ser procesadas se trataban inmediatamente, pero si no, directamente se desechaban y había que meterlas de nuevo si querías realizarlas.

Creo recordar que ni siquiera se molestaban en devolver ese mensaje tan habitual hoy en día y que tan felices nos hace cada vez que lo oímos, en tantos y tantos Centros de Llamadas de tantas empresas de postín... ése de *“Todos nuestros operadores están ocupados, qué pena, penita, pena... haga Vd. el favor de volver a llamar a nuestro simpático 902 dentro de unos minutitos...”*.

Volviendo a nuestro gráfico del *rinconcito*, estos Sistemas especializados en dar servicio a Aplicaciones Transaccionales de Alto Rendimiento requieren colocar la *arena* de nuestro saco de Potencia Integrada del Sistema lo más pegadita posible al eje de los usuarios, para hacer este número lo mayor posible... a cambio de reducir al máximo tanto el acceso a datos de cada transacción como su complejidad.

Lo que resulta es algo parecido a la imagen siguiente.



A mediados de la década de los noventa, con el irresistible auge de internet de aquel tiempo, se comenzó a hablar en serio del *“video-on-demand”*, donde muchos usuarios, personas normales y corrientes como tú, lector, y como yo, podrían conectarse desde sus hogares a un gran servidor de vídeo, sobre todo de películas, y solicitar el visionado de un cierto programa, manejando la emisión como si estuvieras viendo un vídeo VHS como el que entonces teníamos todos: avance rápido, pausa, retroceder, etc.

Ahora cosas así ya existen, se pueden contratar a tu telefónica favorita y funcionan... no exactamente como se suponía que iban a funcionar, pero bueno. Pero en 1997... había que tener imaginación. Se suponía que la capacidad de las líneas telefónicas iba a permitir ese ancho de banda en breve, así que empresas de comunicaciones, de televisión, etc., comenzaron a prepararse para este nuevo mercado y, desde luego, también

la tecnología se preparó.

Comenzaron a aparecer diseños de Sistemas Informáticos (hardware más software) que serían capaces de dar este servicio. Recuerdo entre ellos a Tandem, que anunció una novedosa tecnología de nombre Servernet que permitiría espectaculares rendimientos en proceso paralelo y siempre con la fiabilidad que daba su probada tolerancia a fallos. No tuvo éxito. Poco después Tandem Computers fue comprada por Compaq y nunca llegó a vender muchos sistemas Servernet, si es que vendió alguno, aunque su tecnología básica permanece: es la base del actual Infiniband.

Otros fabricantes comenzaron también a introducir Sistemas de proceso en paralelo; hubo una cierta batalla entre los que abogaban entre sistemas del tipo **SMP** (Procesamiento Simétrico Paralelo), con su arquitectura “shared all”, los de tipo “**cluster**” con arquitectura “shared disk” y los de tipo **MPP** (Procesamiento Múltiple en Paralelo), con arquitectura “shared nothing”.

No voy a entrar a describir las características de cada uno; muchos ríos de tinta hicieron correr en su día y los interesados podéis buscar y leer la muchísima información que hay en la red sobre estos temas.

El caso es que los diversos fabricantes (prácticamente todos) comenzaron a mediados de los noventa la guerra por vender más y más sistemas de procesamiento en paralelo, pero como los Gobiernos y Universidades, los principales clientes de estos Superordenadores, no eran tantos y, además, la mayoría ya tenían una máquina de estas características, el resultado fue que, en la segunda mitad de los noventa, una tras otra, todas las empresas que fabricaban superordenadores quebraron o fueron adquiridas por otras, como es el caso de la propia Cray Research, que fue comprada por Silicon Graphics, o de Thinking Machines, comprada a su vez por Sun Microsystems.

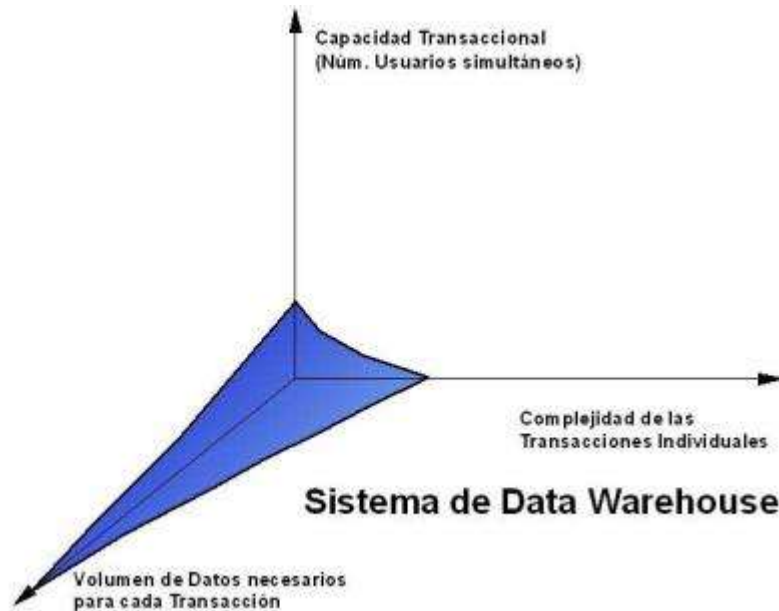
No obstante, a todo el mundo se le ocurrió que algo había que hacer para vender muchas, pero muchas, muchas máquinas de este sofisticado tipo. Y la solución es obvia: **que sean las empresas las que adquieran estos nuevos Sistemas**. Fácil, ¿no?

Pero, claro, las empresas, quitando alguna industrial que fabricara aviones, coches o algo así, no necesitaban grandes ordenadores científicos... ni pequeños, ni ninguno en absoluto. Para pagar la nómina, liquidar cuentas y facturar no son necesarios... Pero sí que es cierto que seguía existiendo la necesidad de poder acceder a la información de negocio de la empresa de forma no estructurada, con “*queries ad hoc*”, y no había ningún “Centro de Información” que valiera para de verdad resolver el problema.

Así que la necesidad de las empresas y de todo tipo de organizaciones, de nosotros los clientes, vaya, de acceder libremente a la información de negocio se unió con la necesidad de los fabricantes de vender para subsistir... y en muy poco tiempo fue *evidente* para todo el mundo que **necesitábamos un Data Warehouse**... aunque en casi todos los casos aún no sabíamos que ése sería su nombre, que de eso sólo nos enteramos algunos años después.

Es claro que para obtener un Sistema Informático orientado a la resolución de esas “*queries ad hoc*” no programadas con anterioridad y sobre información distinta cada vez, se necesitaba distribuir la *arena* en nuestro *rinconcito* de tal modo que se favoreciera la capacidad de acceso a los datos sobre el resto... a costa de tener una complejidad restringida en cada unidad de trabajo y de reducir el número máximo de usuarios simultáneos del Sistema.

El gráfico resultante es, como no podía ser de otro modo, el siguiente:



***“Ponga un Data Warehouse en su vida...”*** se convirtió, alrededor de los años 1996-97, en el *slogan de moda*. Si no tenías ya uno, te tildaban de antiguo o, directamente, de *neanderthal*; si no estabas pensando en construir uno inmediatamente (contratando para ello a la habitual pléyade de consultores, *of course*, a ver si se te iba a ocurrir hacértelo tú solito), es que no te enterabas de nada... una historia que a mí me sonaba bastante... y supongo que, si habéis tenido la paciencia de leerme hasta ahora, también a vosotros debería sonaros.

Veremos en los capítulos siguientes qué es en realidad un Data Warehouse, cómo se vendían y cómo se comenzaron a construir en España en esos años estresantes... comenzaron, digo, porque lo que es “terminaron”, terminaron pocos: muchos de ellos acabaron en sonoros y muy costosos fracasos, porque, para variar, *las cosas no eran ni mucho menos tan sencillas como todo el mundo te las vendía...*



## 18 - El Data Warehouse entró en nuestras vidas...

En el capítulo anterior dejamos la historia del Data Warehouse cuando se produjo la llegada al mercado, al mercado *empresarial*, quiero decir, de todas esas nuevas máquinas de procesamiento masivo en paralelo, casi todas con un cierto *sabor* de UNIX, que herederas de los superordenadores de las décadas anteriores, comenzaron a ser ofrecidas a diestro y siniestro a mediados de los noventa.

Como no había tanto mercado para tantísimo superordenador entre Agencias Sí Gubernamentales, Universidades y Fábricas de Armamento Vario, sus clientes habituales, los fabricantes buscaron un nuevo nicho de mercado donde colocar tanta máquina con decenas de procesadores... Y este hecho coincidió en el tiempo con la creciente necesidad, entre tantas empresas comerciales, de contemplar su información de negocio de forma distinta a la que siempre habían usado: la transaccional.

La evolución de los Sistemas y la creciente competencia en el mercado hacían que cada vez más fuera necesario eliminar trabas a los usuarios en sus consultas de la información procedente del negocio, capturada por los Sistemas Operacionales normales de la Compañía, pero que antes sólo se podía ver de forma agregada, por ejemplo: *Ventas totales en el mes* (sí, pero ¿cómo se distribuyen por negocios, o por tipo de sucursal...?), *Impagados totales en el trimestre* (vale, pero ¿qué tipología de cliente impaga más, solteros o casados; jóvenes o mayores...?), etc.

De acuerdo, todo el mundo vislumbraba que la solución vendría de la mano de esas nuevas máquinas con múltiples procesadores y coste más asequible que un mainframe, pero... ¿cómo conseguimos organizar un Sistema de estos para poder dar el servicio que necesitan las empresas? Porque con tener un hardware potencialmente poderoso no basta, hay que tener además un software que permita realizar este acceso de forma cómoda y eficiente... y de eso aún no había nada a principios de los noventa, aunque pronto iba a cambiar la cosa.

Llegaba por fin el Data Warehouse para solucionarnos la vida... y *aligerar la cartera* de los clientes.

Efectivamente, los fabricantes de máquinas con varios procesadores y habilidad para realizar los procesos en paralelo (cuyo número, además, había crecido bastante en los últimos años 80 y primeros noventa; me refiero aquí tanto al número de modelos como al número de procesadores de cada uno) se volvieron hacia el mercado de las empresas para intentar colocar sus productos, en vista de que su sofisticado mercado tradicional ya no daba más de sí.

Había que crear un nicho que no existía antes. Así que lo primero que hubo que realizar fue una definición formal de **qué era y para qué servía esa cosa nueva**; las empresas tenían una necesidad de acceder a su información de negocio sin cortapisas, pero no estaba claro cómo se podría aprovechar la gran potencia teórica de estos sistemas para obtener resultados tangibles. En una palabra, había que estructurar el mensaje comercial para convencer a las empresas de que les sería muy útil gastarse una fortuna en un nuevo gran sistema, probablemente incompatible con todos los que tenía ya instalados.

Era un trabajo para los especialistas de marketing de los grandes fabricantes... Y estos hicieron su trabajo, ya lo creo que sí.

El resultado fue el concepto de **Data Warehouse**, allá por los años 93-94 del siglo pasado.

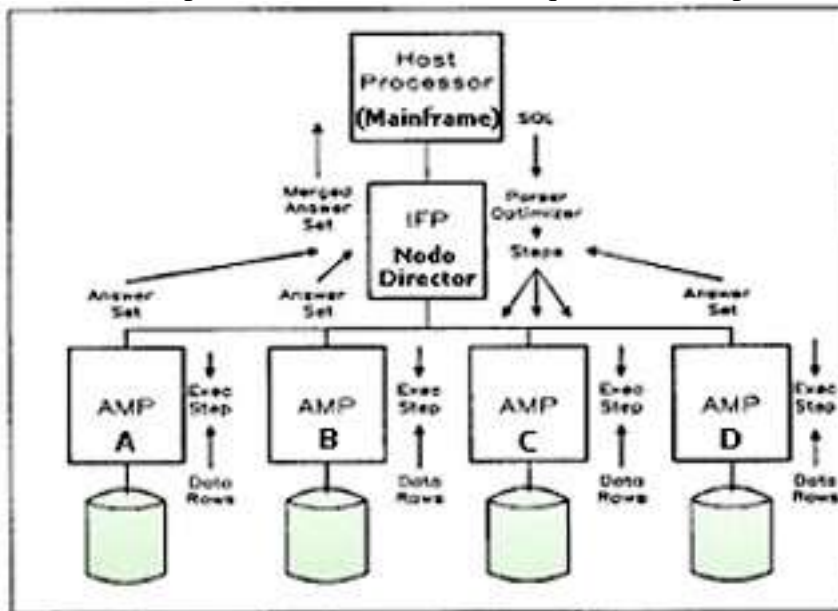
Se habían hecho ya ciertos pinitos en los años 80, sobre todo los de una pequeña compañía californiana llamada **Teradata**, pionera absoluta del concepto de máquina paralela dedicada exclusivamente a la resolución de consultas. Teradata diseñó un Sistema dedicado, donde la clave era su método de interconexión entre procesadores, de nombre YNET, que comunica los diferentes servidores de nombre AMP, por *Access Module Processor*, equipado cada uno de ellos con un barato procesador Intel x86, cada uno de ellos con su propia memoria RAM y su propio disco. Es decir, toda comunicación entre nodos se hacía utilizando el bus (el YNET, pronunciar “*UaiNet*”) así llamado por tener forma de Y, que conectaba dos nodos con uno de jerarquía superior, dando lugar a un interconexión de nodos en forma de árbol binario.

Vale, ya tenían el hardware; pero *¿cómo se gestionaba todo esto para hacerlo realmente eficaz en la resolución de consultas a la información?*

El Sistema Operativo de Teradata estaba basado en UNIX System V modificado para gestionar el paralelismo, pero la clave estaba en su Base de Datos. Ya en 1984, la Base de Datos que llevaba incluida Teradata era Relacional... cuando ¡hasta el año siguiente no estuvo DB2 en el mercado!, aunque sí que estuviera ya en el mercado su antecesor, SQL/DS, para el Sistema Operativo DOS/VSE.

Ya sabéis que al principio de la *vida relacional* hubo muchas reticencias acerca de su rendimiento... y sin embargo los ingenieros de Teradata se dieron cuenta de algo que años después resultó evidente: **Las Bases de Datos Relacionales son perfectas para paralelizar los accesos**, en cualquier caso mucho más que las jerárquicas.

En la imagen se puede observar un esquema de la arquitectura usada por Teradata: su famoso bus YNET (por cierto, se trata de una arquitectura de tipo “shared nothing”).



Explicaré brevemente su funcionamiento con un sencillo ejemplo:

Sea un cierto Sistema de Información que tiene, digamos, diez tablas relacionales, donde la clave primaria de acceso es la misma para cada una de ellas (por ejemplo, el número de cuenta), que debe funcionar en una máquina como la del diagrama, con cuatro

nodos, de nombres A, B, C y D.

Definimos una *clave de particionamiento* al Gestor de la Base de Datos, informándole, por ejemplo, de que los números de cuenta que empiezan entre 00 y 24 están en el nodo A; si empiezan entre 25 y 49, en el nodo B; si entre 50 y 74, en el C; y, por fin, si entre 75 y 99, en el D. Esto es sólo un ejemplo; en realidad lo que hay que procurar es que el volumen de la información asignada a cada nodo sea similar entre ellos, para mantener compensado el Sistema. Si en nuestro caso tenemos muchas cuentas que empiezan por 0 ó 1 y pocas por 8 ó 9, cosa que es lo que suele pasar en los sistemas reales, una clave de particionamiento definida de esta manera sería realmente mala.

También podemos configurar al Gestor para que utilice un algoritmo de *hashing* que distribuya él solito, según sus sofisticados criterios, la información entre nodos. Sólo para que me entendáis, la función hash podría ser algo simple, como dividir el número de cuenta entre 4, que es el número de nodos que tenemos, y usar el resto de la división, de 0 a 3, para designar el nodo que le correspondería a esa cuenta.

Las peticiones (las queries) vienen del “*Host Processor*”, en aquel tiempo casi exclusivamente un mainframe, en forma de un mandato SQL, que es recibido por el IFP (el nodo director). Aquí se compila la query, obteniendo el camino lógico para resolverla en función de las filas involucradas en la query y *cómo están físicamente cargadas* en el sistema, que se envía una vez compilada a cada nodo, de manera descendente, usando para ello la red YNET.

En tiempo de carga masiva de la información, o bien ante una actualización, la lógica del programa director sabe de antemano a qué nodo enviar cada fila, de todas las tablas, dependiendo de su número de cuenta (la clave de particionamiento o de *hashing*).

Pero este tipo de Sistemas está diseñado para ser cargado una única vez y luego consultado muchas veces. Por lo tanto, lo que es realmente importante es que, ante una consulta que, típicamente, llevará un *where*, quizá un par de *joins* o una *subselect*, algún *count* y quizá un *group by*, etc., es decir, la típica consulta *ad hoc* que puede realizar un usuario final buscando información de negocio, **cada nodo es capaz de realizar su parte correspondiente de la consulta, usando su propia memoria RAM, sobre estrictamente los datos que tiene almacenados en su propio disco**. Resuelve joins, cuenta filas, agrupa los resultados... y cuando tiene la información preparada, se la remite de vuelta al nodo director, utilizando en este caso la red YNET de forma ascendente.

Obviamente, todos los nodos pueden funcionar simultáneamente, en paralelo, puesto que, con ese diseño, no necesitan información de ningún otro nodo para resolver su propia parte de la query. Cuando el nodo director ha recibido la información completa de todos los nodos, la consolida, sumando, agregando la información, etc., según el tipo de query de que se trate, y devuelve el resultado por fin al peticionario: inicialmente, al mainframe y, años después, a cualquier sistema principal.

Aún asumiendo que cada nodo es sensiblemente más lento de lo que sería un buen mainframe para resolver cada parte de query, si disponemos de algunas decenas o algunos centenares de nodos, cada cuál resolviendo su pequeña parte en paralelo... el rendimiento es incomparablemente mejor y todo ello con un coste comparativamente reducido, puesto que los procesadores, la memoria RAM, el disco, todos los componentes, en definitiva, son *commodities* normales y corrientes en el mercado y, por tanto, baratos, menos el propio bus de conexión, el YNET en el caso de Teradata, y el software, desde luego, que esos sí que eran caros.

Teradata vendió bastantes sistemas en los años 80 y principios de los noventa,

siendo su venta estrella, a finales de los ochenta, el gigantesco (para la época) sistema que vendió a WalMart, el mayor minorista del mundo, que catapultó sus ventas en todo el mundo. Otros fabricantes siguieron su estela, por ejemplo Red Brick (hoy parte de Informix, o sea, de IBM).

Todos los fabricantes de sistemas UNIX de la época (Digital, Hewlett-Packard, IBM, Sun, etc), pusieron también en el mercado sistemas multiprocesador a lo largo de la primera mitad de los noventa. Todos estos sistemas presentaban arquitecturas diferentes entre sí, componentes distintos y capacidades diversas, pero todos ellos eran de buena calidad y de precio razonable para los tiempos que corrían.

En aquella época se gastó muchísima tinta y saliva en defender las incontestables ventajas y los espantosos inconvenientes de cada tipo de arquitectura, de cada solución, de cada marca... que si el MPP resolvía mejor las queries individuales, que si los clusters eran más adecuados para sistemas distribuidos, que si los SMP's eran definitivamente mejores en todo tiempo y circunstancia...

Ahora sería el momento de escribir dos o tres capítulos completos todos llenos de gráficos y dibujos para explicar todo este lío, las máquinas y arquitecturas que ofertaba cada fabricante, sus pros, sus contras, los argumentos de unos y de otros... pues se trataba de una discusión que, en realidad, era mediática y comercial, aunque disfrazada de puramente tecnológica. Que si IBM vendía el SP2, que si Sun el E10000, Hewlett-Packard el suyo y los demás también... Unos eran MPP, con una arquitectura vanguardista, los otros eran SMP, con un maravilloso bus de conexión... *puffff*.

Veréis, *ya entonces me aburría soberanamente toda esta interminable discusión* sobre si la arquitectura MPP heterodina solucionaría las cargas mixtas mejor que un cluster de SMP's conectados vía satélite... Y si ya me aburría entonces, ¡imaginaos cómo será ahora, cuando han transcurrido ya veinte años...! Así que os voy a ahorrar el rollo (albricias, ¡*por una vez!* pensaréis). Esta vez, y sin que sirva de precedente, ésta es otra historia, pero que la cuente otro.

Y es que, como tantas veces he dicho ya, **lo realmente importante de un Sistema Informático es su software.**

Con la aparición de tanta máquina UNIX multiprocesador todos los fabricantes de Bases de Datos Relacionales (Oracle, Sybase, Informix, DB2...) añadieron capacidades de proceso paralelo dentro de su motor, para aprovechar al máximo el paralelismo que permite la existencia de varios procesadores y diferentes arquitecturas de interconexión. La excepción era el DB2 del mainframe, que tenía capacidad de proceso paralelo desde siempre. De hecho, los mainframes de IBM son de arquitectura SMP, de tipo "shared all", desde hacía muchos años, y DB2 aprovechaba exhaustivamente esta capacidad del hardware y del Sistema Operativo: el MVS, ahora conocido como z/OS.

Todas ellas sabían ya lo que estaba haciendo Teradata, que efectivamente tenía un hardware específico, pero donde la clave de su alto rendimiento radicaba en realidad en su software de acceso a la Base de Datos, así que todas buscaron sus propias soluciones para ser capaces de paralelizar todo lo posible el proceso y, sobre todo, el acceso a los discos, para incrementar el rendimiento de cada query.

Atención: si recordáis los gráficos del *rinconcito* del capítulo anterior, **utilizar paralelismos para resolver una query** (que se supone que debe acceder a muchísimos datos, pues en otro caso *cuesta más el collar que el perro...*), **penaliza, por el otro lado, el**

### **número máximo de usuarios a los que se puede dar servicio simultáneamente.**

Esto es evidente: si tenemos seis procesadores y cuando hay que resolver una query la partimos en trocitos y la entregamos a cada procesador para que resuelva su parte, obtendremos el resultado de esa query particular mucho más rápido que si se encargara un solo procesador de toda ella... pero si cada query se resuelve en un solo procesador, podríamos resolver seis queries simultáneas sin que se molestaran unas a otras... Naturalmente, esto no es *exactamente* así, como seguro que ya sabéis u os figuráis, pero creo que, en principio, es suficiente para comprender el asunto.

**Recapitulemos, pues:** Mediados de los noventa del Siglo XX. Tenemos Sistemas UNIX con capacidad multiprocesador y diversas arquitecturas para proceso paralelo. Existen Bases de Datos Relacionales con capacidad de procesar las queries en paralelo. Las redes y los PC's ya tienen capacidad suficiente como para mover una cierta cantidad de información por las redes y luego tratarla y presentarla localmente. Sabemos dónde están los datos en nuestros Sistema Operacionales, sabemos diseñar y programar el software, tenemos experiencia en el uso de las Bases de Datos Relacionales...

O sea... Ya lo tenemos todo para montar un Sistema de *Queries ad hoc*, es decir, según al usuario de turno le dé la real gana en cada momento, ¿no es cierto?

Ya. Pues eso exactamente es lo que aseguraban y vendían muchos fabricantes y consultores, que, al fin y al cabo, viven de vender productos y proyectos, respectivamente, no de que lo que hayan vendido sirva en realidad para algo. Había muchísimo ruido mediático, congresos, presentaciones, llamadas telefónicas de los comerciales de toda la vida... Mucha presión para los clientes, en definitiva. O sea, como de costumbre. Y hubo algunas empresas, no demasiadas todavía (los *early adopters*, como los llaman pomposamente los consultores; nosotros los llamábamos más bien *los pringaos*) que comenzaron, plenos de expectativas, sus proyectos de construcción de un Data Warehouse que les diera una enorme ventaja competitiva sobre sus competidores y bla, bla, bla.

Estos aguerridos emprendedores hicieron costosos benchmarks entre Sistemas, pruebas exhaustivas, estudios comparativos rigurosísimos... yo estuve personalmente involucrado en varias movidas de éstas. Por fin se compraron un gran servidor con decenas de procesadores y también una base de datos, los que mejor salieron en las interminables pruebas y cuestionarios... o, en su caso, los que decidieron a dedo los directores con motivos más o menos fundados, que de todo hubo. Una vez adquirida toda la infraestructura, comenzaron al fin el proyecto para instalar todo el hardware y el software adquiridos y, a continuación, crear, por fin, su dichoso Data Warehouse.

¿Cómo se haría para construirlo? Yo lo vi; no me tocó a mí liderar uno de aquellos *proto-data-warehouses* de mediados de los noventa (a algunos pobres compañeros míos sí que les tocó), pero sí que sé cómo se atacaron: **con las mismas armas y los mismos bagajes con que se escribían las aplicaciones transaccionales de toda la vida.** En definitiva, *eso* es lo que sabíamos hacer hasta entonces, ¿no?, y no nos había ido tan mal.

Es decir: *Programación pura y dura en Cobol* para extraer los datos; Diseño de la Base de Datos en una perfecta *Tercera Forma Normal*; Programación en *C, C++ o Visual Basic* o lo que fuera, generalmente con ODBC, para acceder al Data Warehouse desde los PCs de los usuarios; un Análisis Funcional de los de siempre, y, por fin, *especificaciones cerradas y firmadas con sangre* por los usuarios.

No era por ser gafe ni me las quiero dar ahora de listo, pero ya entonces a mí me parecía que aquello no iba a funcionar bien, pues me acordaba de mi experiencia personal con el usuario de marketing en mi primer Banco, en la anécdota que relaté en el capítulo

anterior. Incluso tuve el atrevimiento de llevar la contraria a mi jefe en ciertas *Reuniones de Alto Copete*, con lo que conseguí no sólo que no me hicieran caso, sino que además me tildaran de aguafiestas... o algo peor.

Pues, desgraciadamente para casi todo el mundo... tenía yo razón. Y no es que me gustara... es que dicen que *más sabe el Diablo por Viejo que por Diablo*.

**La gran mayoría fracasaron.** Estrepitosamente. Y los que al final consiguieron poner algo en marcha fue con grandes retrasos, menor funcionalidad de la esperada y mucho mayor coste. Y es que no bastaba con tener un servidor, una base de datos, conocimientos en el desarrollo de proyectos tradicionales y buena voluntad. No. **Hacía falta algo más.**

Para poder organizar un Data Warehouse con garantías de que funcione hay que atender a más cosas y, además, utilizar métodos de diseño y programación muy distintos de los normalmente utilizados para los Sistemas de Información que habíamos desarrollado hasta entonces.

Pero, aunque algunos listillos como yo lo intuíamos, pocos sabían cómo atacar correctamente el problema, así que tampoco hay que cargar mucho las tintas en los pobres Jefes de Proyecto que se estrellaron (y perdieron sus jugosos bonos) aquellos años intentando montar un Data Warehouse funcional como buenamente pudieron.

Claro que, al otro lado del charco, había ya algún inteligente y avanzado que estaba poniendo remedio...

Uno de los primeros que ayudó a formalizar el procedimiento de diseño y construcción de un Data Warehouse fue Bill Inmon, que de hecho fue quien acuñó y popularizó el término *Data Warehouse*.

Bill Inmon publicó alrededor de 1992 su libro "*Building the Data Warehouse*", en el que explicó por primera vez las características que debería tener un Sistema de este estilo, que voy (¡cómo no!; si no lo hago, reviento) a resumir a continuación:

**1- Un Data Warehouse debe estar orientado al negocio** (o *Business Oriented*, en inglés, que queda mucho más bonito, dónde va a parar). Es decir, a diferencia de los Sistemas Transaccionales, los de toda la vida, que son *Transaction Oriented*, dado que para ellos lo importante es poder realizar transacciones de forma ágil y eficiente, aquí la información debe almacenarse según la ve y percibe el usuario final, es decir, tal y como está organizada *en el negocio*.

Esto puede parecer una solemne tontería, pero implica cambiar completamente no sólo el método de concepción y diseño del sistema, que ya es mucho, sino la propia forma de pensar de los Analistas, como iremos viendo más adelante.

**2- Un Data Warehouse debe contener datos no volátiles**, es decir, datos históricos en el sentido de que no deben ser modificados nunca. Como si estuvieran grabados en piedra, igual que el milenario Código de Hammurabi que podéis admirar en el Museo del Louvre.

Esto quiere decir, por ejemplo, que si hoy una Oficina realiza un pago de recibo contra una cuenta, pero mañana se dan cuenta de que en realidad el recibo estaba domiciliado en otra cuenta diferente, sería factible, en el Sistema transaccional, anular la operación errónea como si nunca hubiera existido, borrándola literalmente de la Base de Datos, aunque precisamente en el ejemplo que he puesto esto no es o no debería ser posible, pero, para el caso, se entiende, ¿no?.

Sin embargo, si esa operación errónea ha sido ya almacenada en el Data Warehouse,

lo que habría que hacer sería insertar un nuevo registro con la misma operación, pero con el importe negativo: no se debe jamás acceder al registro erróneo para eliminarle. O sea, el Data Warehouse es el reino de la INSERT, nunca de la UPDATE...

**3- Un Data Warehouse debe estar diseñado para proporcionar información a sus usuarios**, en forma de consultas, informes, etc., **nunca para capturarla**: ésta es precisamente la misión de los Sistemas Transaccionales.

Además, estas consultas que debe ser capaz de contestar no son, en general, consultas planificadas: son **queries ad hoc**, es decir, **no es posible predecir cuál es el tipo de pregunta que van a solicitar nuestros usuarios, ni cuándo la van a hacer**, pues de hecho ni ellos mismos lo saben y, por consiguiente, no es posible ajustar a priori el diseño de la tabla, su particionamiento, sus índices, etc., para optimizar el rendimiento o el tiempo de respuesta para ciertos accesos predeterminados, que es la técnica habitualmente utilizada para ajustar los Sistemas Transaccionales.

**4- Además, los tiempos de respuesta de un Data Warehouse deben ser “razonables”**. Eso de *razonable* siempre es objeto de controversia, como se puede imaginar cualquiera. Los usuarios quieren que sean parecidos a los del Sistema Transaccional, o sea, poco menos que instantáneos, mientras los informáticos intentamos convencer a esos mismos usuarios de que dos o tres horas, o diez, para obtener ciertos datos que de otro modo nunca jamás podrían conocer, es una bicoca.

Ambos tenemos razón, claro. Y ésa es la misión del Jefe de Proyecto, si le dejan: unificar criterios, convencer a unos y otros y determinar qué tipo de query es realmente importante que tarde poco y cuál otra no importa que esté resuelta para mañana...

**5- Un Data Warehouse debe estar preparado para gestionar grandes volúmenes de información**, algunos órdenes de magnitud superiores a los tamaños normales en los Sistemas Operacionales.

Un ejemplo: una Compañía de Telecomunicaciones cualquiera de hoy en día puede registrar quizá ciento cincuenta o doscientos millones de llamadas al día... por no hablar de mensajes SMS, mensajes multimedia, tráfico de datos, etc. A poco que esta compañía quiera guardar las llamadas del último año (*en la actualidad se guarda bastante más que eso*, incluso debido a imposiciones legales, ya sabéis: la lucha contra el terrorismo y tal y tal), debe mantener en línea y accesibles por el usuario entre *cincuenta y cien mil millones de llamadas*, es decir, para que se entienda, un uno seguido de once ceros: 100.000.000.000 filas en la tabla de llamadas.

Vale que eso no es nada comparado por el número total de átomos del Universo, pero son una barbaridad de filas para cualquier humilde tabla relacional... Como en estos tiempos de rescates financieros y créditos multimillonarios ya no nos asusta casi ningún numerajo con muchos ceros, igual no os parecen tantas filas... pero yo os aseguro que es una tabla gigantesca, complicadísima de usar, gestionar y actualizar... por no decir de recuperar en caso de error. Y lo digo por experiencia. A mediados de los noventa, con volúmenes medios en las tablas muchísimo más pequeños que ahora, las cifras barajadas eran de quizá de *sólo* diez o quince mil millones de filas, unas *cien o doscientas veces más grande que la tabla más grande de todas las que existían en la instalación en la época*. Realmente son muchos registros, palabrita.

**6- Un Data Warehouse debe tener costes razonables** y aquí hay también desavenencias entre los usuarios, a los que siempre les parece caro, e informáticos, que siempre opinan que es barato... Pero no sólo debe ser razonable el coste del Sistema inicial, el hardware y el software básico, sobre todo la Base de Datos, sino el propio proceso de

Diseño y Construcción, el Mantenimiento posterior, la formación a los desarrolladores y, sobre todo, a los usuarios, etc. En concreto, es vital asegurar que todas las modificaciones posteriores que se hagan al modelo para incluir nuevos datos, cambiar relaciones, añadir nuevas tablas y demás sean baratitas de realizar, porque...

**7- Un Data Warehouse evoluciona mientras evolucionan sus usuarios. ¡Tela marinera!**

Éste es justamente el tipo de Sistema de Información del que los *informáticos de pro* hemos estado huyendo como de la peste durante muchos años: uno cuyo estado vital sea el de las *modificaciones permanentes*. Porque conforme los usuarios van haciendo preguntas y obteniendo respuestas, aunque tarden horas, según van explotando la información contenida en el Sistema también van aprendiendo y haciéndose nuevas preguntas, que muchas veces requieren incorporar cierta información nueva en la que no se había pensado en el diseño inicial... y esto requiere una enorme agilidad para realizar modificaciones en el Diseño de las Tablas, sus índices, etc. y después cargarlas con los datos pertinentes. Aplicar las técnicas usuales de modificación de las tablas en Producción que se aplican en los entornos transaccionales, un procedimiento lento y burocrático que a todos los buenos informáticos nos gusta muchísimo, es una manera segura de certificar el fracaso del proyecto.

En fin: todos nos leímos las teorías de Bill Inmon, las comentábamos entre nosotros, las criticábamos, las medio-usábamos, teníamos el oído atento para intentar aprender de las experiencias de los demás... sólo unos años más tarde llegamos a aprender y *comprender de verdad* lo que significaban, aunque mi opinión sincera es que ni siquiera el propio Bill Inmon se imaginaba por entonces las implicaciones y consecuencias finales de sus *mandamientos*.

Una de esas consecuencias es que el **Sistema físico**, hardware y base de datos, **deben ser independientes y dedicados**. Como vosotros, queridos lectores, ya sabéis cómo son los *rinconcitos de Macluskey* que os expliqué en el capítulo anterior, esto no es ninguna novedad para vosotros, pero en los noventa era muy normal que los responsables del Centro de Cálculo te dijeran: “tengo aquí un Sistema para hacer las facturas que está poco cargado, así que en este mismo Sistema ponemos el Data Warehouse...”.

**Error.** Craso error...

No sólo el Data Warehouse no iba: también la facturación dejaba de ir. La competencia entre procesos por los recursos de la máquina era de tal calibre que ningún proceso acababa bien. Yo viví en mis propias carnes un caso como ése, donde nuestro Cliente tenía su Sistema de Facturación en un Servidor UNIX de mediana potencia, muy descargado durante casi todo el mes excepto cuando se procesaba la facturación; entonces estaba a pleno rendimiento hasta que terminaba.

El Cliente quería ser moderno y tener un Data Warehouse para consultar cositas de las facturas y los clientes y tal, y se empeñó contra nuestra opinión de consultores autorizados en montarlo precisamente en esa máquina que, total, estaba tan *tranquilita*.

Dos semanas después de montarla hubo de adquirir urgentemente otro Servidor dedicado para el Data Warehouse. No sólo tardaban eones las queries que en el servidor de pruebas, con casi la misma carga que el real, iban bastante bien, sino que actualizar una factura se convirtió en un dolor de cabeza, y el proceso de facturación en sí ni os cuento: pasó de durar diez o doce horas, a durar tres o cuatro días... ¡Menos mal que lo habíamos avisado por activa, por pasiva y, sobre todo, *por escrito*, porque si no, nos empitona, pero



bien! Desgraciadamente, otros compañeros no tuvieron tanta suerte en sus proyectos.

Y otra conclusión aún más inquietante de todo esto es que **el equipo humano que debe desarrollar el Data Warehouse debe ser diferente del equipo que hace las aplicaciones transaccionales**. ¡*Tela marinera*, de nuevo!

El Sistema de trabajo, el método de Diseño, la forma de atacar los problemas, etc., son radicalmente diferentes. Por ejemplo, la mayoría de modificaciones al Data Warehouse implican modificaciones en el Diseño de las Tablas, sin tocar para nada la funcionalidad; esto es justamente lo contrario que ocurre normalmente en los Sistemas Operacionales, donde pocas veces hay que modificar Bases de Datos, pero sí, continuamente, la forma de tratarlos.

Y claro, es complicado pedir a una persona que se olvide olímpicamente de los conocimientos adquiridos durante años de trabajo, esos que le han servido para ascender y promocionar en la empresa, y aprender un nuevo sistema, que además es bastante contraintuitivo, pues va en contra de muchas de las premisas atesoradas a lo largo de tantos años de profesión. Desde luego, a mí, que ya sabéis que soy *un genio (je, je)* me costó bastante asimilarlo...

Y hablando de método de Diseño, fue Ralph Kimball quien, en 1997, fijó las bases de diseño del **Modelo Dimensional** que se debería usar al diseñar un Data Warehouse, en su artículo "*A Dimensional Modeling Manifesto*".

El Modelo Dimensional utiliza la misma técnica de representación que el Modelo Entidad-Relación original de Peter Chen, es decir, entidades, relaciones y atributos, lo normal, vaya. Lo que cambia radicalmente es **la forma de entender los datos que deben ser modelados**, la forma de organizarlos en tablas relacionales para construir un Data Warehouse efectivo.

En efecto, para construir un Data Warehouse, hay que grabarse a fuego en la mente que **la información es multidimensional**.

**MULTIDIMENSIONAL**, por si no ha quedado claro.

Multidimensional... Vale. Y... ¿esto qué quiere decir? Pues que de cara a su inclusión en un Data Warehouse, todos los datos, por raros que sean, contengan la información que contengan, pueden ser de dos y sólo de dos tipos: **Métricas y Dimensiones**. Veamos:

**Métricas** (también llamadas **indicadores**) son aquellos datos que representan un valor relacionado con un Hecho de Negocio, con algo relevante para el negocio que ha ocurrido. Son siempre valores numéricos, susceptibles de ser sumados para obtener cualquier valor agregado y responden a la pregunta: ¿**Cuánto...**? Ejemplos: Importe del Adeudo en Cuenta, Importe de Venta, Unidades Vendidas, Minutos de una llamada telefónica, Importe del Siniestro, Número de Hijos...

**Dimensiones** son aquellos datos (generalmente códigos) que **califican o hacen referencia** a ese Hecho de Negocio, cómo se produjo y bajo qué circunstancias, y responden a las preguntas ¿**Quién...**?, ¿**Cuándo...**?, ¿**Dónde...**?, ¿**Cómo...**?, ¿**Qué?**, etc.

Número de Cliente, Fecha, Código de Oficina, Clave de No-Sé-Qué, etc., son todas dimensiones.

Preguntaréis... ¿y qué es eso de un *Hecho de Negocio*?

Pues cualquier Operación que tenga interés para el Negocio: una llamada telefónica que hay que facturar, una venta de un artículo que hay que cobrar, un pago de recibo, un abono en cuenta... cualquier cosa que tenga reflejo contable en una compañía y que sea relevante para el negocio.

Casi siempre un **Hecho de Negocio tiene sólo unas pocas métricas, pero muchas dimensiones**. De ahí el nombre de “Modelo Multidimensional”.

Por si no ha quedado claro todavía: ***Multidimensional***.

Para intentar fijar las ideas usaré como ejemplo una factura muy sencillita (y tanto: por los productos adquiridos, debe ser el ticket de un estudiante) de un conocido hipermercado, para desgranar allí dónde se encuentran las métricas y dónde las dimensiones en semejante hecho de negocio: un sencillo carro de la compra.

En esta factura, como en todas las facturas semejantes de todo supermercado, hipermercado o tienda a secas que se precie, **los hechos de negocio son cada una de las líneas individuales de venta**, es decir, todos y cada uno de los artículos que han sido adquiridos por el cliente en esa operación, en ese carro de la compra.



HIPERCOR, S.A.  
NIF: A-08642668 / Dom. Sect. Mercantil, 192, 28009 - Madrid  
Inscrita en el Registro Mercantil de Madrid,  
T 5245 Gral, Secc 3ª del L. de S. M. 41601, F. 142

### DOCUMENTO DE VENTA



12UIR24YQJFC04DPBL06MFD

Vendedor T.I ExpCent Operac. Fecha Hora EdP12n T.  
65249393 3 0020203 0073578 03/MAR/03 19:20 0168000 00

Descripcion	Cantidad	Importe
AZUCAR	1 B	0,86
ARROZ	1 A	1,53
ACEITE DE OLI	1 B	2,20
ATUN CLACTOLI	2 B	4,80
Precio Unitario	2,40	
CERVEZA RUBIA	1 C	0,27
LECHE SEMI	1 A	0,93
PIZZA FINA	1 B	3,09
CORAZONES	1 B	5,10
CITRICOS UNID	1 A	3,40
SELL. ANCHOA	1 B	0,65
PAN MOLDE	1 B	2,49
YOGUR NATURAL	1 B	2,09
PAPEL HIGIENI	1 C	3,35
DETERGENTE KA	1 C	3,25
AGUA	1 B	0,38
SPAGHETTI	1 B	0,72
TOMATE FRITO	1 B	0,60

**TOTAL COMPRA 35,71**

EFFECTIVO 50,00

CAMBIO EFFECTIVO 14,29

**TOTAL COBRADO 35,71**

Podemos observar, en primer lugar, que la factura está toda ella literalmente llenita de números, pero también que, de todos ellos, la gran mayoría son códigos o dimensiones, mientras que muy pocos de ellos son *métricas* (o indicadores, como más os plazca). Veamos cuáles:

Para cada artículo, existen dos indicadores y sólo dos: el **Número de Artículos Adquiridos**, bajo el epígrafe “Cantidad”, y el **Precio de Venta al Público** de dichos artículos, bajo el epígrafe “Importe”. Indicadores o métricas, ya sabéis que es lo mismo.

Cuando el número de artículos adquirido es más de uno, como en el caso de las latas de atún, se especifica, para mayor claridad, el PVP unitario, que en puridad no es un indicador en sí mismo, sino que sirve para calcular el PVP final.

Y ya no hay más indicadores en esta factura. Ni uno sólo. No los busquéis, que no hay más. Todos los demás datos que aparecen en esta factura son dimensiones. Veámoslas:

El **tipo de IVA** de cada artículo es el código (A, B, C) que va tras la cantidad. Por cierto: ¿os daís cuenta de que el Papel Higiénico y el Detergente pagan el tipo máximo de IVA en España, igual que las cervezas...? Muy curioso: parece que la limpieza sea considerada como un artículo de lujo;

El **código del vendedor**, bajo el epígrafe “Vendedor”, que identifica normalmente a la señorita (o al caballero) que nos atiende en Caja;

El **tipo de terminal**, supongo que es eso, bajo el epígrafe “T.T”, que indicará si la operación se hizo en un terminal de autoservicio, en una línea de cajas, o vete a saber qué otra cosa que seguro que para los responsables del hipermercado tiene interés;

El **Centro Comercial** en que se produjo la Operación, y supongo que también el código de Empresa para algún tipo de control interno que obviamente desconozco, bajo el epígrafe EmpCent. Deduzco que el “002” será el “código de empresa” y el “0988” hará referencia a una determinada tienda donde se produjo efectivamente la venta;

El **código de Operación**, bajo el epígrafe “Operac.”, en el que seguramente viene el número de terminal más un identificador secuencial de la operación. En banca, cuyos entresijos conozco mucho mejor, se hace exactamente igual;

La **Fecha y Hora** en que se produjo la Operación, bajo los epígrafes “Fecha” y “Hora”; y **dos códigos más** que no se me ocurre qué pueden indicar, pero que seguro que algún valor tienen.

Por fin; tras el Total Compra, que no es más que la suma de los importes de los artículos, aparece una nueva dimensión: La **Forma de Pago**, en este caso “*En Efectivo*”, porque si hubiera sido hecho el pago con tarjeta de crédito, aparecería su número, y, de alguna manera, la indicación “con tarjeta de crédito”, “de débito”, etc., en el ticket, así como el código de autorización y demás datos de la operación de crédito.

Por cierto, el Total de la Operación, el TOTAL COMPRA, no es una métrica básica, sino una *métrica derivada*, obtenida mediante suma de los precios de los diferentes artículos adquiridos en la misma operación. Métricas derivadas hay muchas; por ejemplo, podía haber otra más que aparece mucho en facturas de éstas, aunque precisamente en ésta no: el Número Total de Artículos adquiridos, que también se obtiene con una simple suma de la “Cantidad” de Artículos del ticket.

Otras métricas derivadas podrían ser: IVA de cada artículo (el tipo vigente de IVA aplicado al PVP), porcentajes de participación (cuánto representa la venta de la región de Cantabria respecto del total nacional), de variación (incremento de ventas entre un trimestre respecto el mismo trimestre del año anterior), de consolidación (ventas agregadas de las Secciones de Limpieza y Perfumería, por ejemplo) y muchos más tipos que seguro que se os ocurren.

Además, en esa factura sencillita hay más dimensiones que no aparecen físicamente en la misma, pero que *estar*, están: por ejemplo, cada artículo lleva su código, el código de barras que, aunque no es mostrado en la factura (en su lugar prefieren mostrar sólo el nombre del artículo), evidentemente sí que es conocido por la tienda y debidamente almacenado; este código asociará indisolublemente el producto vendido a una determinada sección (Limpieza, Ultramarinos, Congelados...), que tendrá un responsable de su venta, su reposición, etc.; cada artículo tendrá un determinado Proveedor, que a su vez será de una cierta región o país, que repondrá la mercancía vendida de una determinada manera (por pedido, automáticamente, por venta, etc).

En cuanto al vendedor, éste estará asignado a una determinada tienda y departamento, tendrá una cierta categoría laboral, un determinado tipo de contrato, una cierta edad, un sexo, un curriculum... Los terminales de tal tipo serán de una cierta marca y modelo y tendrán un cierto historial de averías, de uso, etc.

Y así con todo. Existen lo menos doce o quince dimensiones asociadas a algo tan simple como un ticket de venta de un supermercado...

Bien, pues lo importante es que **todos estos Hechos de Negocio representan una imagen del negocio completo de la Compañía**. Es decir, si agrupamos todos los Hechos de Negocio individuales (las operaciones de venta, las devoluciones, los descuentos... todos) en una sola tabla, que llamaremos la **Tabla de Hechos** (*Fact Table*, en inglés) y mantenemos un periodo temporal importante (dependiendo del negocio, un año, dos, tres... o toda la vida), **por mera acumulación de sus indicadores podremos conocer cualquier información relacionada con el Negocio de la Compañía sobre cualquier ámbito dimensional**.

Parece una obviedad, sí, pero esto es importantísimo, **esta característica es la que da su valor real a los Data Warehouses**. En las aplicaciones transaccionales de toda la vida se definían algunos valores acumulados representativos del negocio, se precalculaban, casi siempre en batch, y se permitía consultarles a voluntad. Por ejemplo, la venta mensual por región, la venta acumulada por sección y centro, la venta en efectivo y con tarjeta de crédito, etc... Pero no es bastante, porque ¿qué pasaría si en un momento dado necesitábamos conocer el importe de venta en efectivo de un determinado centro en ciertas horas concretas de los fines de semana? Esto no es nada extraño. De vez en cuándo, y cada vez con mayor frecuencia, es necesario conocer este tipo de información que nunca antes de este momento se había necesitado, bueno, o *sí* se había necesitado, pero nunca se había podido obtener.

Pues lo que pasa es que *no teníamos la menor idea, ni siquiera la posibilidad de obtener la información* como no fuera haciendo costosos programas específicos que se usarían para calcular el dato *en el futuro*, cuando por fin se pongan los programas en marcha, porque el pasado muerto... muerto está, y el dato perdido no hay quien lo recupere.

¿Cómo se representa gráficamente esta realidad según el modelo dimensional definido por Ralph Kimball? Pues mediante una representación **en Estrella** (Star Schema, en inglés) o, mejor aún, con una representación **en Copo de Nieve** (Snowflake Schema).

En realidad, una u otra representación son lo mismo, sólo que en el modelo en estrella las dimensiones son de únicamente un nivel, mientras que en el modelo en Copo de Nieve, las dimensiones pueden tener jerarquías, que es lo que ocurre en virtualmente todos los casos. Una jerarquía dimensional representa subagrupaciones dentro de la dimensión; por ejemplo, un año tiene la buena costumbre de dividirse en doce meses, que a su vez se dividen en días, estos en horas, éstas en minutos, etc. Si se mantiene esa jerarquía, es posible comparar las ventas de los cinco primeros días de febrero de los últimos tres años, o las de los últimos sábados de junio, por franjas horarias, o las de ciertas secciones de Alimentación, por medio de pago... y otras muchas más que seguro que, si les damos la oportunidad, se les ocurrirán a los responsables del negocio.

En el centro del diagrama se encuentra siempre la Tabla de Hechos, que suele ser de gran tamaño. Y cuando digo *gran tamaño*, aquí sí que me refiero a **GRAN TAMAÑO**. Miles de millones de registros (o billones de registros, como dirían los americanos), en la

mayoría de los casos.

Y alrededor de la Tabla de Hechos, los rayos de la estrella o las hermosas formaciones fractales de los copos de nieve, donde están recogidas las tablas dimensionales, muchísimo más pequeñas en tamaño, que explican cada uno de los Hechos de Negocio contenidos en la *tabla gorda*.

En la siguiente ilustración está representado un modelo en Copo de Nieve (reducidísimo) de los datos de llamadas de voz de una compañía telefónica cualquiera. Los hechos de negocio son precisamente las llamadas, con unos pocos indicadores cada una (minutos y segundos de duración, precio, coste y poco más), mientras que alrededor habrá muchas dimensiones. En el caso de una *Telco*, fácilmente pueden llegar a ser catorce o quince.

En la figura se han representado solamente unas pocas de todas estas dimensiones:



**La Dimensión Temporal**, con el Día y la Hora concreta en que se produce la llamada;

**La Dimensión Cliente** (lo que antes llamaban “El Abonado”), con la representación de su localización geográfica. En la práctica sería mucho más complejo de lo que aparece representado en la figura, algo así como Central Telefónica, Distrito Postal, Ciudad, Provincia, Comunidad Autónoma, País, Región, etc., pero es que, además, tras el Cliente aparecen muchísimas dimensiones más: *sexo, estado civil, antigüedad como cliente, servicios contratados, actividad del cliente*, etc; y

**La Dimensión Operador de Destino**, donde se marca cuál fue el Operador del teléfono que recibe la llamada, que puede ser la misma compañía u otra de la competencia.

Hay muchas más dimensiones en este telefónico modelo: Cliente de Destino (sólo si la llamada es a un Cliente de la misma Operadora), tipo de llamada, tipo de tarifa, tipo de

central telefónica... En fin, creo que podéis haceros una idea.

Solamente cuando por fin aprendimos esta forma novedosa de *Comprender El Mundo Multidimensional* fuimos capaces de construir Data Warehouses efectivos, es decir, que de verdad funcionaran, y esto ocurrió a partir de 1999 o así... aunque todavía hubo coletazos de espectaculares fracasos hasta bien entrados los dosmiles...

A ello (a que se pudieran construir de forma efectiva, quiero decir) ayudaron una serie de herramientas software que fueron, y siguen siendo, cruciales en la gestión y explotación de los Sistemas de Data Warehouse.

De todas ellas hablaré en el siguiente capítulo, donde, por obra y gracia de los expertos del marketing informático, veremos cómo ocurrió la asombrosa transformación del Data Warehouse en *Business Intelligence*.

## 19 - Y el Data Warehouse se convirtió en... Business Intelligence

Hemos visto en capítulos anteriores, en primer lugar, cómo fueron apareciendo y siendo progresivamente ofrecidas al mercado empresarial las máquinas de *proceso masivo en paralelo*, y después, sobre todo, los fundamentos teóricos necesarios para poder construir Data Warehouses con éxito, que fuimos aprendiendo sobre la marcha... y a fuerza de golpes, todo sea dicho.

Vimos allí cómo gracias al trabajo de Bill Inmon, que definió las características que debía tener un Sistema de este tipo, incluso inventando el propio término “Data Warehouse”, y al de Ralph Kimball, que fue quien formalizó el Diseño de un Data Warehouse mediante Modelos Dimensionales, dando origen a Modelos en Copo de Nieve como el de la ilustración, fueron decisivos para que los pobres mortales pudiéramos construir los primeros Data Warehouses... o al menos intentarlo, porque la mayoría de estos primeros Sistemas acabaron, a pesar de todo, fracasando.

Efectivamente, las cifras de fracaso que graciosamente nos daban las consultoras, sobre todo ésas que *de todo hablan y nada solucionan*, eran abrumadoras: más del 80% de los proyectos de Data Warehouse iniciados aquellos años, en la segunda mitad de la década de los noventa, o se cancelaban sin implantar nada o, en el caso de que sí que acabaran con un sistema funcionando, era con mucha menos funcionalidad de la prevista y a mucho mayor coste del inicialmente planeado.

El mensaje comercial evidente era que si no les contratábamos a ellos, *precisamente a ellos*, para construir nuestro Data Warehouse... el trompazo estaba garantizado. Claro que, aunque les contrataras a ellos, *precisamente a ellos*, también tenías ciertas “posibilidades no nulas” de pegarte el mismo trompazo, o peor.

En una palabra, **no sabíamos aún cómo demonios construir un Data Warehouse y terminar el proyecto en tiempo y coste y con la funcionalidad esperada...** Ya lo he dicho.

Como esto lo sabíamos hacer ya (bueno, *más o menos*) en los proyectos de construcción de aplicaciones tradicionales, es decir, online, batch, en fin, los Sistemas Operacionales de toda la vida, pues nos frustraba. Nos frustraba a nosotros, los informáticos... y les frustraba enormemente a nuestros Usuarios y/o Clientes, con toda la razón. De cómo evolucionó el asunto hasta convertir la construcción de un Data Warehouse en algo *trivial y sencillo* como es hoy en día (bueno, *más o menos trivial y sencillo*) hablaré en este capítulo. Repito lo que he dicho ya muchas veces: esto no es una historia oficial de nada, es un extracto de mis recuerdos y sensaciones durante esos primeros y vertiginosos años de los comienzos del Data Warehouse, época que viví trabajando en una consultora buenísima de ésas que, entre otras muchas cosas, vendía todos los proyectos de este jaez que podía.

Os preguntaréis... ¿Por qué fracasaba tanto proyecto de construcción de Data Warehouses en todos los países y en todos los sectores?



Bueno, no sé exactamente por qué fracasaban todos los que lo hicieron, y fueron muchos, pero sí de algunos que ocurrieron aquí en España de los que fueron involuntarios protagonistas algunos compañeros míos y de algunos otros de los que fueron protagonistas otros profesionales pero que por diversos motivos llegué a conocer bastante bien. Creo que me puedo permitir el lujo de extrapolar las razones principales de esos fracasos (total, lo llevo haciendo todo el tiempo...) con bastante probabilidad de que este diagnóstico sea lo suficientemente correcto y compartirlas aquí con vosotros, queridos lectores.

En mi opinión, la principal causa de los fiascos fue no tanto la falta de estándares o teoría, sino **el propio desconocimiento que teníamos los informáticos sobre cómo enfrentarse a estos proyectos.**

Me explico: La construcción de los primeros Data Warehouses se le encargó generalmente a magníficos profesionales, con muchos años de experiencia en el desarrollo de proyectos informáticos y en la gestión de proyectos complicados. La razón era evidente: en aquel tiempo se trataba de los *proyectos estrella* en muchas empresas y también para muchos fabricantes y consultores: las máquinas eran muy costosas, pues aunque lo fueran menos que un buen mainframe, aún costaban un cerro de millones y precisaban de muchos servicios especializados y, por tanto, caros. Así que los suministradores eligieron para dirigir estos proyectos a los mejores profesionales que tenían disponibles para intentar garantizar en la medida de lo posible su éxito.

Loable intento. Pero *contraproducente*, aunque, desde luego, ninguno de nosotros lo podíamos siquiera sospechar en la época, y yo tampoco. Y ahora os preguntaréis... Pero, vamos a ver, si los suministradores y los propios clientes utilizaron para estos proyectos a la flor y la nata de sus profesionales... *¿Dónde está el problema, pues?*

Pues el problema está en que estos excelentes profesionales, entre los que creo que yo mismo me contaba, atacaron el proyecto de la misma forma en que habían atacado exitosamente tanto proyecto complicado con anterioridad... y **ése no es el método adecuado** para atacar la realización de estos proyectos. Veamos por qué.

En primer lugar, se gastaron muchos meses en intentar *definir con precisión los requerimientos de negocio* que debía cumplir el Sistema... **Y no había forma de fijarlos.** Cambiaban continuamente; cuando parecía que por fin se había llegado a un consenso definitivo sobre ellos, aparecían nuevas necesidades no contadas... era *el proyecto de nunca acabar*.

Esto acabó con la paciencia y la salud de bastantes Clientes, Usuarios y, sobre todo, de Jefes de Proyecto que, simplemente, no entendían cómo era posible que un Usuario no supiera qué era lo que necesitaba para su trabajo, qué información concreta habría que suministrarle en el futuro. Estaban acostumbrados a diseñar y construir Sistemas Operacionales muy complejos, pero en todos ellos había quedado relativamente claro desde el principio cuál era su objetivo y sus requerimientos fundamentales.

**¿Sí? Pues aquí, no.**

Un día parecía que lo más importante que debía contemplar el Sistema era el acceso a la información de venta desglosada *ad infinitum*, pero, al día siguiente, eran los ratios por sucursal el punto clave que de verdad, de verdad, debería resolver el sistema... para cambiar a cualquier otra cosa una semana después, y la otra, y otra más. Los pobres Jefes de Proyecto no entendían nada... ¡y los Usuarios, a su vez, no entendían cómo era posible que tan selectos (y caros) profesionales no les entendieran *a ellos!*

La razón de todos estos malentendidos seguramente la sabéis ya: **Es imposible que un Usuario sepa qué aspecto del Negocio necesitará verificar o qué tipo de**

**información necesitará mañana para responder a las cambiantes necesidades de las compañías en el competitivo mundo en que vivimos.** Ni mucho menos el mes que viene o el año que viene... Sólo que esto lo sabemos *ahora*, no entonces. Cada cuál se encastilló en sus posiciones... y casi todos los proyectos acabaron mal, como no podía ser de otro modo.

Todo esto sería, ya de por sí, más que suficiente para condenar al fracaso a multitud de proyectos... **Pero aún hay más**, que tiene que ver esta vez no tanto con el método, aunque un poco sí, sino con el propio uso de la tecnología en sí. Hay que tener en cuenta que casi todos los proyectos de construcción de Data Warehouses que se vendieron a mediados y finales de los noventa fueron vendidos, bien por fabricantes (del carísimo hardware, mayormente), o bien por consultores de todo pelaje, que se apoyaban en todo lo que tenía que ver con la tecnología en los propios fabricantes, del hardware y de la Base de Datos, aportando ellos la mayor parte de los servicios profesionales: Diseño, Desarrollo, Dirección del Proyecto...

De esta manera, todos los planteamientos de proyectos que vi aquellos años hacían un enorme, disparatado, casi único énfasis en las **sublimes capacidades de las Bases de Datos y de las máquinas que las soportaban** (si revisamos ahora esas “sublimes” capacidades de entonces y las comparamos con las “normalitas” de ahora, nos daría ciertamente la risa, pero entonces eran... eso, *sublimes*) y obviaban todo lo demás. Todo el resto que hiciera falta, todo lo demás, ya se solucionaría más adelante, *ya inventaremos algo*, a base de servicios profesionales a mansalva (*programando*, para que se me entienda) o buscando algún productito por ahí para solucionar algún tema puntual, como por ejemplo para acceder a los datos y otras tonterías parecidas, que no eran importantes en absoluto, puesto que lo realmente importante era el *Recojoservidor* y la *SuperBase de Datos Paralela*... que, tontamente, juntos suponían quizá el 85% del coste total del proyecto, mire Vd. qué casualidad. En fin.

En una palabra, se magnificaba el papel del Servidor y de la Base de Datos y se minimizaba y trivializaba todo lo que hay alrededor.

Pues resulta que **justo esas menudencias que hay alrededor del Servidor y su Base de Datos definen lo que es verdaderamente importante para que un Sistema de Data Warehouse funcione.**

Me explico nuevamente: Para tener un Data Warehouse funcionando es preciso un Servidor dedicado, de cuanta más capacidad, mejor, y una buena Base de Datos con capacidad de ejecutar procesos en paralelo, configurada, por cierto, de una forma bastante distinta de la que es óptima en un Sistema Operacional, pero ésa es otra historia y será contada en otro momento.

Ahora bien, esa Base de Datos, para que valga para algo, debe ser, antes que nada, cargada con los datos precisos... datos que han sido capturados en algún momento por los Sistemas Operacionales y que están allí, tan ricamente guardados en algún sitio, organizados para dar servicio a esas aplicaciones transaccionales y, por tanto, son *transaction oriented*. Como en el Data Warehouse los datos deben ser *business oriented*, ya lo dijo Mr. Inmon, no queda más remedio que extraerlos de los Operacionales, cambiarlos de formato, transformarlos, acumularlos o desagregarlos, según el caso, y cargarlos después con su nuevo formato escrupulosamente orientado al negocio en el Data Warehouse.

Porque resulta que *no sirven tal y como están en los Sistemas Transaccionales*. No hay caso: **No Sirven**. Punto.

En esos primeros proyectos, en todos sin excepción, esta parte de extracción,

transformación y carga de los datos fue sistemáticamente minusvalorada. Se pensaban los Jefes de Proyecto, Analistas, etc., que con unos cuantos programas o SQL's bien paridos sería suficiente para resolver el problema y que ya se abordaría el problema con los medios tradicionales, o sea, programando, cuando llegara el momento. No lo sabíamos entonces, desde luego, pero resulta que **resolver esta fase cuesta alrededor del 80% de todo el tiempo que cuesta el proyecto.**

Esto quizá merezca una breve explicación, porque es posible que os extrañe esta cifra tan elevada. El diseño fundamental de cualquier Sistema de Data Warehouse, como vimos en el capítulo anterior, es el de una única y gigantesca Tabla de Hechos y una serie de Dimensiones explicativas, más pequeñas. En la tabla de hechos se representan, juntas, todas las métricas (o indicadores) valiosos para el negocio.

Por ejemplo, tratándose de un normalito Sistema de Venta, el típico de cualquier gran empresa de distribución, hipermercado o similar, la tabla de hechos contendrá todas las operaciones individuales de venta y las devoluciones que se hayan producido en un cierto periodo temporal: un año, dos o más, según. Esta tabla de hechos contiene dos métricas evidentes: el número de unidades adquiridas y el importe, que se pueden encontrar fácilmente en la aplicación actual de venta, pues siempre van juntos.

Supongamos que, además de la venta pura y dura, queremos ir algo más allá y conocer cosas sobre, por ejemplo, la rentabilidad que obtenemos con nuestros productos o nuestros suministradores. Quizá ése sea, precisamente, uno de los motivos clave que han llevado a la compañía a gastarse una fortuna para crear su Data Warehouse. Para ello bastaría con introducir una nueva métrica en la tabla de hechos: el **precio de coste del producto** que hemos vendido ese día a esa hora. De este modo es sencillo averiguar no sólo lo que hemos vendido, sino con qué margen bruto se ha realizado cada venta. Fácil, ¿no?

**Pero hay un problema con esto:** ningún minorista que yo conozca tiene estos datos en la misma aplicación que los datos de venta; están en otra aplicación: la de Compras, la de Stock, la de Contabilidad, no sé, pero nunca en la de Venta, siempre en otras tablas diferentes y con ciclos de creación de la información distintos. O sea, hay que añadir a los datos de venta, que provienen directamente de los terminales de punto de venta, datos de otra aplicación que tiene una problemática diferente y, por qué no, codificación diferente, dado que las reglas de negocio seguidas en el diseño de uno y otro sistema pueden ser muy distintas... y de hecho, por lo poco que yo sé, normalmente lo son.

Es decir, que añadir una simple métrica a nuestra tabla de hechos, una métrica *evidente* desde el punto de vista del negocio (eso es justamente lo que quiere decir eso de *business oriented*) puede ser una pesadilla desde el punto de vista de la realización informática.

Quizá los productos, por ejemplo una tableta de chocolate, estén codificados de forma diferente, con su EAN en la Aplicación de Venta, para facilitar su contabilización, pero con el código del fabricante en la de Compras, porque así se facilitan los pedidos; quizá la tableta de chocolate que estamos vendiendo en el híper de Torrevieja se la podemos comprar físicamente a diferentes proveedores, cada uno de ellos con sus propias condiciones de precios, y sea difícil determinar quién nos sirvió el producto que estamos realmente vendiendo ese día a esa hora; es posible que tabletas de chocolate servidas en diferentes momentos a lo largo de semanas o meses lo hayan sido a diferentes precios, incluso por el mismo fabricante (a veces los precios cambian, ¿sabéis?), por lo que no sabemos qué coste concreto asignar a cada tableta vendida; es muy habitual que las cadenas de hipermercados tengan acuerdos con sus proveedores por los que, según las cantidades

que se consigan vender a lo largo del año, se apliquen a posteriori descuentos adicionales por parte del proveedor, lo que en el sector llaman “rappel”: ese nombre debe ser porque es difícil de averiguar a priori cuál será ese rappel finalmente, así que los de Compras necesitan de un *adivino* para poder estimarlo...

Y muchos más problemas adicionales que os ahorro. ¡Y todo, para introducir una *simple, escuálida y sencilla a la par que evidente métrica* en nuestra tabla de hechos! Espero haber sido capaz de daros idea, con este sencillo ejemplo, del alcance real del problema.

Y claro, todo esto se hacía inicialmente a base de escribir programas al modo usual de los días: Análisis Funcional, Diseño Técnico, Programación, Pruebas... Un berenjenal de mucho cuidado. Y fue siempre muy, muy minusvalorado en tiempo y coste en aquellos primeros proyectos.

**El choque con la dura realidad se llevó por delante multitud de proyectos** que no fueron capaces de solventar la problemática con que se encontraron. Y los pocos que, sorprendentemente, lo consiguieron, se toparon de bruces con el mantenimiento de la aplicación: cada vez que se necesitaba un dato nuevo había que bucear por la maraña de programas para, primero, localizar dónde rayos estaba ese nuevo dato en las tablas del Sistema Transaccional y luego integrarlo en lo existente... lo que era mucho, pero que mucho trabajo. El tiempo de respuesta ante una nueva petición era enorme, igual que el de modificar una aplicación transaccional, puesto que en definitiva estaban hechas de la misma manera.

Todo esto de por sí era tremendo... Pero es que ahí no queda la cosa: **aún hay más**. Suponiendo que milagrosamente todo esto fuera realizado convenientemente y en plazo y coste, y que la información fluya sin problemas desde nuestros Sistemas Operacionales hasta el Data Warehouse de alguna mágica manera, aún quedaba otro problema espinoso que resolver: el acceso de los usuarios a los datos.

Todo este dinero gastado, todo este montaje, en definitiva, lo hacíamos para dar acceso a los usuarios o clientes a sus datos y permitirles obtener información acerca de su negocio como nunca antes lo habían podido hacer. Maravilloso. Pero es que, para poder hacer esto, de algún modo esos usuarios finales deberían ser capaces de solicitar al Sistema la información concreta que deseaban obtener, el Sistema debería enterarse de la petición, traducirla a instrucciones comprensibles por las Bases de Datos, o sea, SQL, obtener los datos y devolvérselos al peticionario de una forma que éste pudiera comprender y usar.

Proyectos hubo que laboriosamente consiguieron cargar las tablas del Data Warehouse con la información solicitada... **y no hubo luego quien pudiera acceder a ella**.

Quizá se suponía por los responsables del proyecto que los usuarios deberían aprender SQL y lanzar sus queries, quizá desde Excel, Access o algo así, igual que en el mainframe se hacía con el QMF... y que encima las hicieran correctamente, con todos sus joins bien puestos...

**Pues no, otra vez no.** Pocos usuarios finales sabían SQL y menos aún entre ellos sabían cómo montar las complicadas queries necesarias, con sus join, subselect, group by, etc. Aquellos pocos que eran capaces de hacerlo aún debían recoger las filas en bruto que devuelve una query *a pelo* y manejarlas de alguna manera para que sus resultados fueran visibles.

Y los que no sabían... esos no tenían el menor interés en aprender SQL, lo cual es

perfectamente lógico: a ellos les pagan por obtener información del negocio, para tomar decisiones en función de esta información y para mejorarlo, no para programar nada. Claro que quizá se pensaba programar una *sencilla* aplicación cliente/servidor, en Visual Basic o C, o lo que fuera, para que enviara su query y representara los datos obtenidos *en bonito*. Pero, desde luego, para que esto pudiera ser viable, no se trataría para nada de una *aplicación sencilla*, sino de una de las más complicadas que se podían escribir en la época.

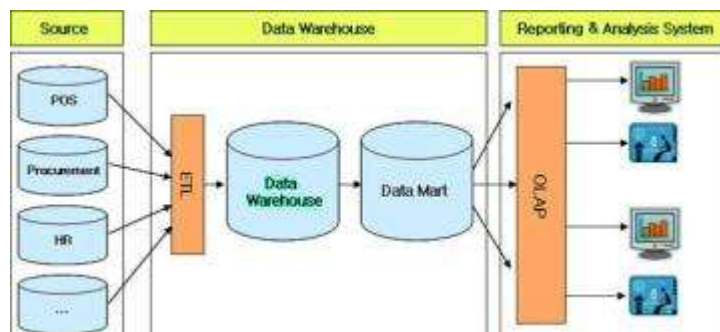
Como veis, también este aspecto fue clarísimamente minusvalorado por los vendedores de estos primeros Data Warehouses que, según recuerdo, basaban su mensaje comercial en la potencia nominal de sus máquinas y la excelencia de las Bases de Datos para tratar montañas de datos, pero descuidaron en sus ofertas tanto la entrada de datos al sistema como luego la salida.

Si me permitís la analogía, es como que alguien te venda un extraordinario amplificador de sonido de *audiófilo*, con lámparas de vacío, una sensibilidad estratosférica y un precio en consonancia... y que luego este amplificador maravilloso no tenga clavijas para poder conectar ninguna fuente de música, ni tampoco enchufes para conectar altavoces... o sí que los tiene, pero te venden como entrada un micrófono monoaural para que tomes el sonido del altavoz de tu radio portátil a transistores y, como altavoces, unos buenísimos de cartón que te daba pena tirar y has reciclado de tu viejo Seat Panda...

En fin, naturalmente que aprendimos de todos estos fracasos. Algunos los sufrieron en carne propia y les costó dinero y salud... y los demás que, por pura suerte, nos escapamos, procuramos escarmentar en cabeza ajena... dentro de lo posible.

¿Cuáles fueron esas medidas que fuimos tomando paulatinamente? Unas tuvieron que ver con las herramientas software a utilizar, otras con el método y otras, por fin, con la propia forma de concebir un Data Warehouse... Vayamos por partes.

Antes de nada, veamos cuál es el esquema de funcionamiento de un Data Warehouse, que es, con variantes más comerciales que otra cosa (con o sin *Staging Area*, con o sin *ODS*, etc.), el que puede observarse en el gráfico a continuación. Los datos de origen están en los diferentes Sistemas Operacionales, los que están representados a la izquierda del esquema. Mediante ciertos programas, se **Extraen** de allí, se **Transforman** para hacerlos coherentes unos con otros y por fin se **Cargan** en las Bases de Datos que componen el Data Warehouse. De ahí lo de “**ETL**” (*Extract, Transform and Load*) del dibujo.



Una vez cargado, se puede replicar parte de la información para cargarla en otras tablas más pequeñas y especializadas, incluso en otras máquinas diferentes, orientadas generalmente a un Área o Departamento concreto de la empresa (Finanzas, Contabilidad, Marketing, etc.), tablas que mantienen una vista parcial del Data Warehouse. A estos

“mini-Data-Warehouses” se les conoce como **Data Marts** y suelen ser más pequeños en tamaño y más manejables que los Data Warehouses de los que proceden... aunque yo conozco empresas que tienen Data Marts de 8 ó 10 Teras. Todo es relativo, como podéis comprobar.

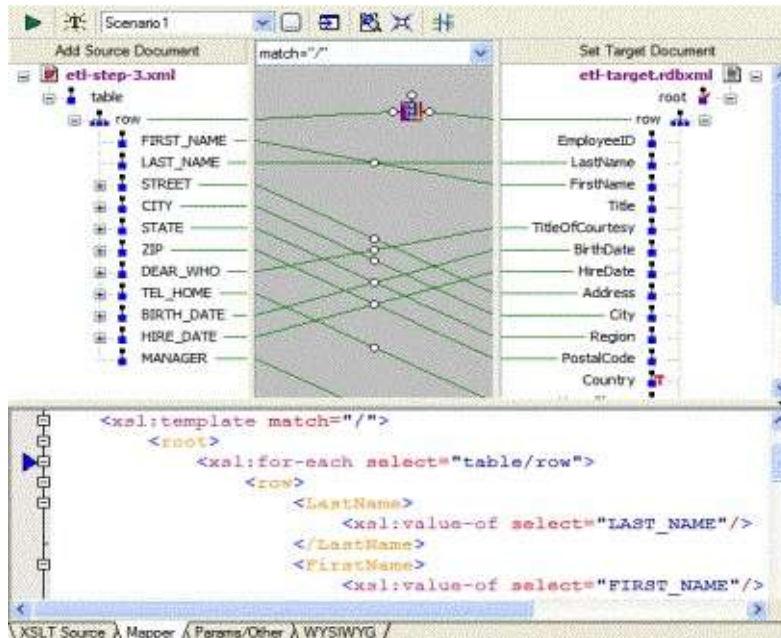
Por fin, los sufridos usuarios, a la derecha del todo, acceden a los datos del uno o de los otros, en función de su conveniencia, bien para obtener información sobre algo que preguntan (Query&Reporting), donde se consigue como salida un informe o un gráfico con la información solicitada, bien para navegar por la información contenida allí, mediante la técnica conocida como OLAP, de *OnLine Analytical Processing*, que es como el *online de toda la vida* (OLTP), pero sobre la información de negocio contenida en los Data Warehouses o Data Marts. La idea es que, una vez obtenido cierto informe en el que se detecta algo interesante en una o varias líneas del mismo, se pueda navegar jerárquicamente a lo largo de las diferentes dimensiones para obtener más detalles de ese interesante hecho encontrado hasta poder determinar sus causas últimas.

La primera vez que vi este gráfico, que con ligeros cambios usan todos los fabricantes y consultores, pensé que era algo obvio, pero que la dificultad estaba en dar a la cosa el dinamismo que requiere. Si para resolver cada parte del diagrama nos ponemos a programar como sabemos (o, al menos, *como sabíamos*, no estoy yo muy seguro de que ahora *sigamos sabiendo*), estos sistemas jamás llegarían a buen puerto.

En primer lugar, la industria comenzó a dar soluciones para el proceso de Extracción, Transformación y Carga de la información, **ETL** en sus siglas inglesas, para hacerlo más sencillo y, sobre todo, facilitar el mantenimiento. Algunos se dieron cuenta de que los programas que había que hacer para estos menesteres eran casi siempre del tipo *Sota-Caballo-y-Rey*, o sea, muy parecidos entre sí, así que se les ocurrió que se podía automatizar en todo o en parte ese proceso. Tenían razón.

El proceso normal de carga de datos en el Data Warehouse implica, conocidas las estructuras de las tablas de entrada y las de salida, trazar el camino que deben seguir los datos en su viaje desde los sistemas operacionales hasta el Data Warehouse. Así, buena parte del proceso podría describirse de manera visual, identificando cómo calcular, uno a uno, cada dato contenido en las tablas de salida, es decir, las que forman el Data Warehouse, describiendo cuál es su origen, qué cálculos o transformaciones hay que realizar, etc. ¿Recordáis que ya Jackson había especificado en su Método de Diseño de Sistemas JSD que *el diseño debería comenzar a partir de las salidas*? Lo expliqué como mejor pude y supe en el capítulo dedicado a las Metodologías de Desarrollo. Pues esto es más de lo mismo.

Imaginemos, pues, una pantalla como la que hay a continuación. A la derecha están todos los campos de las tablas que componen el Data Warehouse, que se obtienen directamente del catálogo, las copias o de donde sea y, a la izquierda, todas las tablas que serán origen de datos en el Sistema Operacional.



Ahora vamos tomando campo a campo de la tabla de salida y vamos encontrando en las tablas de entrada los campos de dónde proceden. Muchos vendrán sin cambio alguno: Código de Cliente, Nombre, Dirección, etc., se traspasarán de uno a otro sistema sin cambio alguno, con un movimiento simple. Otros sufrirán algún tipo de proceso. Por ejemplo, para calcular el número de cuentas que posee un cliente se especificará que se deben contar las ocurrencias de cada Código de Cliente en la tabla de Cuentas. Algunos serán simples cálculos elementales: el porcentaje de recibos devueltos de ese Cliente se calculará en base a los recibos devueltos sobre los totales, mientras que obtener otros valores requerirá de cálculos complejos... y así con todo.

Naturalmente, siempre habrá algún campo cuyo endemoniado proceso de obtención no esté entre las posibilidades standard de los productos; en esos casos, se prepara una *EXIT* con el código necesario, programado *ad hoc* para la ocasión... pero deberían ser los menos casos y, afortunadamente, lo son.

Todos estos productos almacenan las definiciones que visualmente se introducen en un cierto fichero (son los *metadatos* o datos sobre los datos; ya hablé de ellos cuando hablaba de las herramientas CASE) y, una vez terminado de definir cada campo de cada tabla de nuestro Data Warehouse, se ejecuta un proceso que genera, por un lado, los programas, en el lenguaje adecuado, que deberían servir para extraer los datos del servidor origen, por ejemplo, Cobol/DB2, si se trata de un mainframe, que es lo normal en muchos casos; por otro, los programas para realizar la consolidación y los cálculos intermedios, la transformación, que pueden ejecutarse en la plataforma origen o en la de destino, incluso en una distinta, a conveniencia del diseñador; y, por fin, los programas para cargar la información resultante en las tablas, las de hechos o las dimensionales, del Data Warehouse, por ejemplo, en C con Oracle, si se trata de un servidor UNIX con Oracle.

Además, generan los scripts necesarios para ejecutar dichos programas en sus entornos correspondientes. Luego estos programas se compilan, exactamente igual que si los hubiese escrito nuestro programador favorito, se usan los scripts... *et voilà*, ¡nuestro Data Warehouse cargado en un pis-pas! Bueno, en *dos* pis-pases o tres...

La gran ventaja es evidente: que los programas funcionan. *Siempre y siempre a la primera*. Y además, como la definición se encuentra almacenada en los metadatos, si se incluye un nuevo campo en el Data Warehouse o debe calcularse alguno de una forma diferente, basta con especificar cómo debe ser cargado y, al generar nuevamente el código completo, queda integrado con el resto de forma inmediata y fiable.

Eso sí, puede ocurrir que, ante un cambio aparentemente menor, el pequeño demonio que el producto lleva dentro decida resolverlo con un proceso completamente distinto al que hacía hasta entonces... para desconcierto de diseñadores y, sobre todo, del departamento de Operación... Pero funciona, *siempre* funciona.

El primer producto ETL en salir al mercado fue Prism, seguido al poco tiempo por ETI Extract y Carleton; todos ellos seguían este esquema de generar programas que debían ser compilados para ser ejecutados. Prism es el abuelo que está en la base de uno de los productos ETL más exitosos de la actualidad, DataStage de IBM; ETI Extract sigue existiendo malamente hoy en día y de los de Carleton no se encuentra nada de ellos en parte alguna... cosas que pasan de forma habitual cuando de productos antiguos, o no tan antiguos, se trata.

Algunos años después comenzaron a aparecer otros productos diferentes que, una vez hecha la definición de forma similar, hacían el movimiento de los datos en tiempo real sin necesidad de generar, compilar y ejecutar programas. Es decir, una vez cargados los metadatos de forma similar a la descrita antes, lo que se hace es arrancar unos procesos residentes en ambos lados (el mainframe y el servidor UNIX, para que me entendáis), procesos coordinados entre sí que realizan el trabajo interpretando el contenido del *metadata* y, además, de forma desasistida. Planificada, eso sí, pero desasistida.

Su ventaja: es mucho más sencillo de operar, al no requerir los pasos adicionales de compilación, creación de scripts, etc. Y es más sencillo conseguir que paralelicen el proceso si tenemos problemas de ventana batch, a cambio de requerir muchísimo más consumo de CPU y recursos que en el otro caso.

*DataStage*, finalmente adquirido por IBM, y *Powercenter*, de Informatica (sí, hay una compañía norteamericana denominada “Informatica”, sin acento; muy creativos con el nombre no lo son, no), son algunos productos ETL de este tipo, pero ahora hay muchos, pero muchos, así...

Volviendo al Siglo pasado, mal que bien teníamos resuelta ya la parte más problemática, el movimiento de datos, de *muchos* datos entre sistemas, modificando, transformando y cargando de forma razonablemente efectiva estos datos en nuestro Data Warehouse. Pero... ¿qué pasaba con el acceso a la información?

Pues también había compañías que estaban dando soluciones para facilitar el acceso a los datos de un Data Warehouse por parte de sus usuarios, es decir, permitirles diseñar sus informes, ejecutarlos, incluso navegar por ellos sin necesidad de saber programar nada, ni tampoco del apoyo del Departamento de Informática. Hubo varios pioneros, algunos de los cuales ya no existen, pero los que mayor presencia comercial tuvieron en España esos años, a finales de los noventa, fueron:

**Business Objects**, una compañía francesa pero que rápidamente dio el salto al otro lado del charco y que fue probablemente la primera compañía del mundo en entrar en estas lides. Especializada en *Query&Reporting*, o sea, realizar informes de calidad de forma sencilla y con un buen aspecto visual, aunque con escasa capacidad de navegación, se convirtió en seguida en el líder mundial por licencias vendidas.



En 2007 Business Objects fue adquirida por SAP. De hecho, SAP ofrecía desde hacía años Business Objects como la herramienta principal para acceder a su “SAP Information Warehouse”, el *pseudo-Data-Warehouse* que SAP ofrecía, de nombre *SAP Business Information Warehouse*, que supongo que sigue ofreciendo con el mismo u otro nombre para explotar la información contenida en sus tripas...

**Cognos**, una compañía canadiense que cubría con su producto básicamente el mismo espectro de Query&Reporting que Business Objects; de hecho fue su mayor competencia esos años, entre otros muchos más productos. En 2008 fue adquirida por IBM... para mí que ésta fue una adquisición forzada para intentar compensar la adquisición de Business Objects por SAP un año antes y así competir en todos los frentes con la compañía alemana. Ignoro qué tal les irá ahora a ambos.

**Microstrategy**, una compañía estadounidense que, a diferencia de los anteriores, mostraba su fuerza en la navegación a través de las dimensiones, usando muy eficientemente la técnica del *drill-down*, en lo que se vino a llamar ROLAP, una más de las *tropecientas* siglas que aparecieron aquellos años... Sus informes eran ciertamente más feos que los de la competencia y sus gráficos correctos, aunque bastante espartanos, pero su capacidad de generar un excelente SQL para realizar la navegación le hizo posicionarse claramente como el líder indiscutible de este segmento.

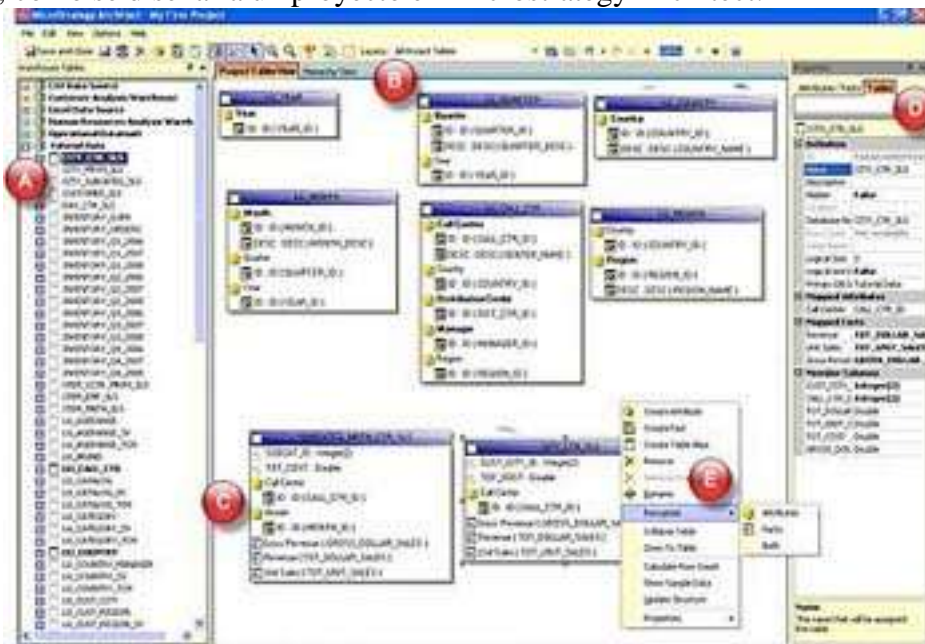
Y, con una concepción radicalmente distinta, pero con un gran éxito, la también estadounidense **SAS Institute**, originalmente especializada en el tratamiento estadístico y matemático de los datos. De hecho, “SAS” quiere decir *Statistical Analysis System*, o sea, que el nombre lo dice todo, aunque parece que últimamente han eliminado toda referencia a lo de “Statistical” y “Analysis”... otra compañía más que reniega de su propia historia, ¡qué asco! A lo largo de los años fue SAS incorporando a su corazón, su extraordinario motor estadístico, un caparazón de software que permite acceder a los datos, manipularlos, gestionarlos, procesarlos e imprimirlos con gran agilidad, obteniendo, casi sin quererlo, un potente lenguaje de cuarta generación... que es en realidad el que le ha reportado la mayoría de las ventas desde entonces.

Aunque no era exactamente igual que los anteriores, pues exigía una cierta programación (bueno, no: *programación en toda regla*) por parte de los usuarios para obtener sus informes, tuvo bastante éxito como herramienta de acceso a los Data Warehouses de aquellos años.

Por lo que yo sé, la mayoría de estos productos tienen un origen similar: Una Compañía de consultoría tradicional es contratada por una empresa para que haga no sé qué Sistema de Información, y en esa compañía hay, entre varios otros, dos tipos cruciales: el *ingeniero listo*, al que se le ocurre que, en vez de programar seiscientos informes todos parecidos pero todos diferentes, si escriben un generador de informes se quitan el problema de una tacada (y, además, ¡lo escribe y funciona!, el tío) y un *visionario del negocio* que se da cuenta de que esa cosa se puede vender bien y monta una compañía para empaquetar ese mismo software que ya tienen hecho y venderlo... Por ejemplo, en Microstrategy el gurú es Sanju Bansal, un indio (de la India, quiero decir), un genio de la tecnología que fue quien parió el motor de generación de SQL, y Mike Saylor, el listo visionario y genio del marketing...

Muchas de estas iniciativas fracasaron antes o después y, tras la inevitable “*consolidación del mercado*”, la forma eufemística de llamar al baile de adquisiciones de los últimos años, casi no quedan ya compañías independientes en este mercado.

En la práctica, todos estos productos solucionan el acceso por los usuarios a la información contenida en el Data Warehouse de forma parecida. En la ilustración se ve, por ejemplo, cómo se diseñaría un proyecto en Microstrategy Architect:



**1** El Diseñador define la información del proyecto contenida en el Data Warehouse, con su modelo en estrella o en copo de nieve, especificando las tablas físicas con sus atributos y asociándolos a los conceptos de negocio que necesitará el usuario para pedir la información. Así, se le informa al sistema de que la tabla ATY07RWD, por ejemplo, es en realidad la Tabla de Hechos del Sistema, donde el atributo CODOFIWK es el código de oficina... Como también le diremos que la tabla ATY08RGL es la que representa la dimensión “Oficina”, donde el atributo CODOFIWK es el código de oficina y NOMOFIWK, por ejemplo, es la denominación de esa oficina, ahora el Sistema sabe que, si le pedimos alguna información de la tablas de hechos, como por ejemplo los saldos totales de un tipo de producto agrupados por oficina, basta con hacer un join entre ambas tablas usando para realizarlo los campos que contienen el código de oficina en cada tabla y poder así añadir el nombre de la oficina, además del código, en el informe resultante.

**2** Terminada esta definición básica, el producto la almacena internamente en algún sitio, generalmente en tablas relacionales en el propio sistema donde reside el Data Warehouse: son sus metadatos (o Universos, según la nomenclatura usada por Business Objects). Ahora es el momento de definir las métricas derivadas, como porcentajes de incremento o de participación, fórmulas aritméticas, etc., que quedan almacenadas también en los metadatos.

Se definen también filtros o condiciones a aplicar a los datos, agregaciones... toda esta información queda asimismo almacenada en los metadatos. Lo bueno de toda esta fase es que, si se han diseñado bien las tablas, se trata de un proceso muy rápido: en una semana o así se puede tener completamente modelizado un Data Warehouse de tamaño medio, y eso es muy rápido para nuestras ancestrales costumbres.

**3** Ahora, cuando el usuario desea una cierta información, selecciona qué métricas

desea, con qué dimensiones y con qué condiciones, y ejecuta el informe.

El producto genera el SQL que envía al Data Warehouse, probablemente vía ODBC, SQL que se supone que está optimizado para la Base de Datos concreta donde están los datos... *se supone*, al menos, y cuando recibe la respuesta, la formatea para dejarla con la pinta solicitada por el usuario y se la muestra por fin.

Así de fácil... bueno, *o no*.

Por ejemplo, si el usuario desea conocer la Venta por Sección de todos los Supermercados ubicados en Madrid, selecciona un sólo indicador: “Importe de Venta”, dos datos dimensionales: “Nombre de Sección” y “Nombre de Supermercado”, y define un único filtro: que sean Supermercados de la Región “Madrid”. Lanza el informe para que se ejecute... y ya está. Un procedimiento rápido, sencillo e intuitivo que, como resultado, obtiene y pone a disposición del usuario un informe comprensible, perfectamente utilizable para realizar su trabajo.

Además, el producto de marras nos permite luego navegar dimensionalmente, por ejemplo, seleccionando la Sección “Pescado” y pidiendo al sistema que nos muestre ahora la venta de los departamentos de la Sección de Pescado (Lubinas, Doradas, Meros...) de los Supermercados de Madrid... y así hasta poder alcanzar las operaciones de venta individuales, si fuera preciso.

Todas estas herramientas eran ya entonces de gran calidad (ahora son, obviamente, de *más calidad y mejor tecnología*, hacen más cosas, mejor y... *más caro*), y la decisión de compra de una u otra tenía que ver más con las filias o fobias de cada uno que con su utilidad real. Mi recomendación de consultor, igual que las de mis compañeros, era que se debería comprar la herramienta adecuada a la función que el cliente deseaba... ¡aunque fueran dos distintas! Claro que, a cambio, nosotros, los consultores, cobrábamos comisión de todas ellas, *je, je*.

Era una situación bastante normal que hubiera que dar servicio a una serie de usuarios *livianos*, que consultarían básicamente informes preparados de antemano cambiando, eso sí, las condiciones, los filtros, en función de sus necesidades, mientras que otros usuarios mucho más preparados necesitarían hurgar en las tripas del negocio para encontrar información relevante... Business Objects o Cognos, por ejemplo, eran perfectos para los primeros, mientras que Microstrategy, incluso SAS, lo eran para los del segundo tipo.

Pero esto, que era muy evidente para casi todos los que conocíamos de cerca la tecnología, casi nunca lo era nada para el Director de Informática, que prefería mil veces tener un solo proveedor en vez de dos. Esto, además, era muy normal en la Administración Pública (Ministerios, Organismos, Ayuntamientos...), cuando publicaban un concurso en el que sólo cabría una única herramienta de cada clase... Y me constan algunos fracasos precisamente por seleccionar la herramienta inadecuada; además, luego nadie daba su brazo a torcer y los comerciales de la herramienta seleccionada aseguraban que si las cosas no iban bien, era debido a que nosotros, los humanos, no éramos capaces de hacer que su *maravilloso* software funcionara como era debido. Incluso en algunas ocasiones... ¡hasta tenían razón!

En España unas compañías vendieron más que otras, debido no sólo a la tecnología, sino también al soporte postventa, es decir, a la calidad de los profesionales de cada empresa para poner a funcionar su producto en los clientes reales, formar a los responsables de cada empresa, dar soporte en caso de problemas... y ser flexibles ante la inevitable

*negociación con el cuchillo en la boca* que acontecía siempre con los responsables de pagar el proyecto. Y es que estos proyectos eran, aquellos años de finales del Siglo XX, bastante caros.

Ya teníamos estupendos servidores con muchos procesadores y capacidad de proceso en paralelo, magníficas Bases de Datos que eran capaces de sacar partido de estas máquinas sofisticadas, así como las técnicas de modelización dimensional requeridas, teníamos también el software que nos ayudaría a realizar la extracción de los datos, su transformación y carga en el destino y, por fin, las herramientas de acceso a la información por el usuario final... Ahora, **ahora sí** que podíamos construir por fin Data Warehouses de excelente rendimiento, acabando el proyecto en tiempo y con los costes presupuestados...

*Eeh...* Tras todo lo que lleváis leído, creo que imagináis quizá la respuesta: **Pues no.**

Todavía nos faltaba encontrar un par de puntos definitivos, los detalles finales un tanto sorprendentes que nos permitirían, por fin, poder comenzar a construir Data Warehouses con celeridad, bajo coste y expectativas cumplidas... pero encontrarlos, identificarlos y, sobre todo, interiorizarlos, nos costó aún algún batacazo más.

En primer lugar, e **importantísimo** (pero *importantísimo*), es que **nunca, nunca, jamás se le debe preguntar a un usuario “qué información desea consultar”** (o “qué información necesita”, es decir, los famosos *Requerimientos de Negocio* imposibles de fijar), sino **de qué información dispone**. Atentos, porque esto que acabo de contaros no lo oiréis ni leeréis seguramente en parte alguna. Es un secreto inconfesable de la profesión... Esta técnica de no-pregunta elimina de un plumazo cualquier interminable discusión sobre requerimientos que no hay forma de cerrar, ahorrando muchísimo tiempo... y problemas; ahora es muchísimo más sencillo quemar esa etapa.

Ante la inevitable pregunta de los usuarios de *¿y qué información podré obtener?*, la respuesta es: **TODA**, toda la que tengas: Tomaremos TODA la información contenida en tus Sistemas Operacionales, toda ella sin excepción, la modelizaremos, la extraeremos, la cargaremos en el Data Warehouse y allí, si lo hemos hecho bien, que esto es otra historia y será contada en otro momento, **cualquier pregunta es posible**: cualquier cruce de información, cualquier agregación a cualquier nivel, cualquier métrica derivada...

Unas consultas tardarán poco, otras mucho o muchísimo... aquellas que tarden en exceso se aligerarán mediante las diferentes técnicas usuales en Data Warehouse (creación de tablas agregadas, cambios en los índices, partición de tablas, etc., técnicas que no voy a detallar aquí), y listo.

Me he cansado de deciros que el usuario no sabe qué información necesitará mañana, es más, es que ¡no puede saberlo! *¿Para qué vamos, entonces, a preguntarle lo que desea, si no lo sabe?* ¡No le preguntamos y ya está! Y no penséis que esto es tan raro: cuando se está seleccionando un ERP o un producto para llevar la contabilidad, las nóminas, etc., nadie pregunta por los requerimientos de negocio, que están clarísimos, pues son los propios de la contabilidad, la nómina...

No os podéis hacer una idea de lo que engrasa el desarrollo de los proyectos este sutil (bueno, vale, *no tan sutil*) cambio en la orientación del proyecto de construcción de un Data Warehouse.

Y el segundo punto crucial es que **el diseño debe hacerse pensando en la idiosincrasia y las necesidades de la herramienta utilizada para acceder a la**

**información;** en caso de tener más de una, de la más restrictiva (ésa era, casi siempre, Microstrategy). Cuando tanto el diseño de la Base de Datos como la nomenclatura de métricas, atributos, índices, etc., se adaptaba como un guante a las necesidades de esta herramienta... el proyecto iba bien y funcionaba a la primera... bueno, *a la segunda*, que a la primera ya sabéis que no funciona nunca nada.

Recuerdo un colega al que le tocó en suerte dirigir el tercer o cuarto intento de poner a funcionar el Data Warehouse de un gran Banco español. Los anteriores habían acabado en sonoros fracasos y habían costado la defenestración sucesiva de la Consultora de turno. Coincidimos en una de las mil presentaciones, congresos o charlas que había por doquier, me contó su problema, o mejor, su *patata caliente*, y se me iluminó el cacumen para darle un solo consejo: “*Ficha a un auténtico experto en Microstrategy*, pues era ésa la herramienta que había seleccionado aquel Banco, *págale lo que pida y deja que sea él quien haga el diseño físico de las tablas*... ¡y no permitas que metan las narices en el diseño los especialistas de Base de Datos del Banco o de tu empresa, o tendrás problemas!”.

Varios meses después coincidimos de nuevo en otro evento. **El hombre estaba feliz como una perdiz:** aunque había tenido poco menos que llegar a las manos con los responsables de administrar las Bases de Datos, al final se salió con la suya... ¡y el proyecto iba viento en popa! De hecho, que yo sepa ese banco sigue explotando el mismo Data Warehouse, corregido y aumentado a lo largo del tiempo pero con el mismo diseño básico de entonces. ¡Por una vez, mis consejos habían servido de algo! Mi colega fue promovido en su empresa, su salario, incrementado y, pocos años después, fue ignominiosamente despedido... pero ésa es otra historia y será contada en otro momento.

Además, simultáneamente a todo esto y a las cien movidas más que ocurrían todas ellas simultáneamente y, como de costumbre, liándolo todo, también se estaba promocionando por los gurús de *la cosa* otra tecnología similar... o completamente diferente, según se mire, aunque, en todo caso, relacionada: el **Data Mining**.

Curioso nombre: “*Data Mining*”...

Consiste esto del Data Mining en “realizar descubrimientos de cierta información no trivial oculta dentro de los datos”, utilizando para ello bien algoritmos estadísticos, bien provenientes de la inteligencia artificial. En una palabra, la información se extrae a base mayormente de pico y pala (de ahí lo de *Minería de Datos*), *pico y pala figurados*, pero pico y pala, al fin. Éste será el tema concreto al que estará dedicado el próximo capítulo, el último de los dedicados a la explotación de los datos de nuestros Sistemas por los Usuarios, así que no digo aquí nada más.

Bueno, pues el caso es que entonces los expertos de marketing volvieron a hacer su trabajo. Ya sabéis que de tanto en cuando se les exige que inventen un nuevo término para denominar lo mismo que antes tenía otro nombre, por aquello de la novedad, y ésta fue una más de esas ocasiones. Había que buscar un término marketiniano poderoso, pegadizo, que permitiera aumentar las ventas de toda esta serie de productos y los jugosos servicios que conllevaban... y el término finalmente elegido fue: **Business Intelligence**.

Bajo el paraguas del Business Intelligence caben todo tipo de productos hardware, software, de diseño, de acceso a la información, de seguridad, de control... vamos, que tiene tantos componentes (¡o más!) que el resto de disciplinas informáticas. Es uno de los *leit motiv básicos* de venta de las consultoras actuales, los fabricantes de hardware y

servicios, las compañías de servicios profesionales... es *el último pico por escalar* para informatizar las empresas hasta el último nivel y, queridos lectores, ya sabéis lo que pasa en estas circunstancias, porque ha pasado muchas veces antes: “Si no pones un Business Intelligence en tu vida, eres... ¡eres un *neanderthal*!”). Aún se podría hablar unos pocos centenares más de páginas sobre estos apasionantes temas, de la máxima actualidad en nuestros azarosos tiempos, pero voy a dejarlo aquí. Repito: hay muchísima información disponible y no creo que mis incontinentes palabras puedan ya aportar mucho más.

*¿Mi conclusión?* Igual queréis escucharla...

Después de unos comienzos titubeantes, grandes fracasos y algunos éxitos, mucha jerga nueva y muchas tecnologías enfrentadas a muerte unas con otras (por ejemplo: MOLAP vs. ROLAP vs. HOLAP, de lo que no he hablado nada, pero que hizo correr ríos de tinta en su día; por cierto, en nuestros tiempos ya quedó claro que la tecnología ganadora de las tres es la ROLAP), tras muchos fracasos y algunos éxitos, repito, poco a poco los éxitos fueron siendo mayoría... y yo creo que ahora es difícil fallar en un *proyecto normal* de creación de un Data Warehouse. Realmente es más sencillo de diseñar y de construir que un Sistema Transaccional y, en general, tienen rendimientos aceptables a costes razonables y una gran utilidad.

Y esto ha llevado a que se usen en muchísimas ocasiones de forma espuria, falsa, no natural: como meros emisores de los informes o las consultas que ya no se programan en los Sistemas Online normales y corrientes. Es decir: se escribe una aplicación tradicional, con sus actualizaciones, sus informes preimpresos y sus cartas, sus altas, bajas y modificaciones... pero luego sólo se codifican unas pocas consultas básicas, porque resulta más barato construir un Data Warehouse y permitir que los usuarios hagan allí las consultas que les dé la real gana sin molestarnos a nosotros, los informáticos, que así podemos dedicarnos a cumplir escrupulosamente nuestro *ISO-doscientos-millones* de cada día... No, no era para *eso* para lo que se pensaron estos sistemas, ni tampoco era para eso para lo que tantos pasamos tantos sudores, pero, al menos, si funcionan y la gente los usa porque sirven para algo, bienvenidos sean. El mundo está llenito de sistemas estupendos que no sirven para nada.

Lo que sí que ocurre es que todas, *todas* las grandes empresas y, por supuesto, todas las Agencias Sí Gubernamentales de todos los países del mundo mundial tienen Data Warehouses de tropecientos Teras de tamaño... o *Petas*... o *Exas*, donde almacenan todo lo que acontece en su negocio, con sus productos, sus empleados, sobre sus clientes... en definitiva, todo lo que saben sobre nosotros. En la duda, hoy en día se guarda todo. Pero *todo, todo*. No sabes qué será lo que el día de mañana te hará falta, así que lo mejor es guardar hasta el último retazo de información que caiga en tus manos. Un director de una de esas gigantescas agencias de información manifestaba que, para encontrar una aguja en un pajar, lo primero que tenías que tener era el pajar. Luego, con tiempo, ya encontrarás la aguja... ¡Bienvenidos al **BIG DATA**!, que ya tenemos un nuevo palabro para llamar de otro modo a lo mismo de siempre...

El caso es que **lo saben todo sobre nosotros**. O, al menos, *lo podrían saber*. Saben por qué páginas navegamos, qué nos descargamos, lo que pagamos por ello y cómo lo hacemos, lo que comemos y lo que vestimos, a quién llamamos por teléfono y lo que le contamos, cuánto dinero debemos y a quién, dónde vamos de vacaciones y por qué medios, nuestros hábitos diurnos y nocturnos...

Pero que no cunda el pánico... aún.

El que la información, Teras y Teras de información, esté almacenada en algún gigantesco ordenador en algún antiguo silo de misiles nucleares no quiere decir que *de verdad sepan mucho*... O, mejor, que *ya* sepan mucho. Como bien decía Johann Wolfgang von Goethe a principios del siglo XIX, lo importante no es tener grandes cantidades de información a nuestra disposición, sino poder aprovecharla para aumentar nuestro conocimiento... y eso está todavía lejos, por más que las técnicas de Data Mining, de las que hablaré en el próximo capítulo, hayan *adelantado* como *las ciencias* en La Verbena de la Paloma: ¡*una barbaridad!*

Hasta que no despierte el *Tecnonúcleo* de Hyperion, de Dan Simmons, los humildes mortales podemos estar tranquilos... o eso espero... No. En realidad, eso *anhelo*.

## 20 - El descubrimiento de la Minería... pero de Datos

Hemos visto hasta ahora cómo los Sistemas de “Business Intelligence” eran (y son) utilizados para generar informes variopintos sobre cosas sucedidas en el devenir de la empresa, los famosos Hechos de Negocio. Los usuarios finales solicitan cualquier información que se les ocurre sobre la marcha en base a las necesidades de cada momento, o sea, lo que en la jerga se llaman “*queries ad hoc*”, y el Sistema responde con la información solicitada en un tiempo razonable, es decir, segundos, minutos, horas, días... dependiendo tanto de la complejidad de la consulta como de su prioridad. Todo esto está muy bien y abre a los usuarios de negocio una enorme ventana a la información que tenemos almacenada en nuestros sistemas. Pero no es bastante... **nunca es bastante**. Qué forma de ser la nuestra, que nunca estamos conformes con lo que tenemos: ¡siempre queremos más!

No se trata sólo de obtener información en completos listados con bonitas gráficas... se trata de que esa información sirva para algo. Se trata, en definitiva, de **convertir la Información en Conocimiento**. Es ese conocimiento el que permite a los gestores de la empresa interiorizar situaciones, sacar conclusiones y, por fin, tomar las decisiones adecuadas para mejorar el comportamiento de la empresa. O así debería ser, al menos.

En una palabra, no sólo hay que poder consultar la información y navegar por ella... Hay que encontrar las íntimas relaciones entre los datos que explican muchos hechos de negocio, buscar nichos desconocidos o no explotados para generar nuevas oportunidades o mejorar las existentes...

De todo eso tratará este capítulo, de todas esas técnicas que reciben el muy comercial y, desde mi modesto punto de vista, poco apropiado nombre de **Data Mining**.

Repito una vez más: esto no es ninguna crónica de nada. Contaré lo que yo he visto, aprendido e inferido de mis escasas intervenciones en proyectos de Data Mining; existe muchísima información sobre el tema en la red, pues es uno de los puntos calientes de la profesión, y lo que yo cuente aquí no tiene por qué coincidir con la historia oficial... ¡qué se le va a hacer! Ya lo decía Eduardo Marquina, en su obra teatral “En Flandes se ha puesto el sol”: *España y yo somos así, señora...*

Efectivamente, no cabe la menor duda de que pedir cualquier información que se nos ocurra a nuestro Data Warehouse y que éste nos conteste de forma fiable y en un tiempo adecuado es una enorme ventaja competitiva... pero no basta. Un ejemplo muy, muy conocido (aunque falso: **es una leyenda urbana como una casa**, pero sirve para ejemplarizar el tema) servirá para fijar las ideas:

En los años 80, WalMart, la mayor compañía de distribución minorista del mundo de entonces y de ahora, compró e instaló uno de los primeros Teradata, de los que hablé hace un par de capítulos, para averiguar cosas acerca de su negocio. Y al cabo de poco tiempo empezó a circular por doquier la historia siguiente:

*“WalMart había descubierto que los sábados por la tarde se producía un interesante hecho de negocio: resulta que tenían ese día muchos clientes que compraban pañales y cerveza simultáneamente...”*. Había versiones que iban más allá y aseguraban que



esos clientes eran, además, hombres... de sexo masculino, vaya.

La consecuencia que se sacaba es que WalMart había identificado un enormemente valioso *nicho de mercado* donde había un conjunto considerable de personas, u hombres, según la versión, que compraban simultáneamente dos productos aparentemente muy poco relacionados entre sí: pañales (para ponérselos a los bebés, supongo) y cerveza (para bebérsela ellos, me imagino, no los bebés).

Y WalMart había sacado mucho partido a tal conocimiento... porque además, en la historieta se interpretaba la razón de tal comportamiento: hombres con hijos en edad de llevar pañales, sin posibilidad de salir por la noche al tener que cuidar a su hijito, se proveen en abundancia de cerveza para alegrarse la noche viendo seguramente con algunos amigos un partido de algún esotérico deporte por la tele.

Bueno, esto no ocurrió jamás. *Es una leyenda urbana*, como antes dije. Pero durante muchos años nos estuvieron martilleando docenas de consultores (no exagero, de veras) con el maravilloso ejemplo y las posibilidades que encierra poder descubrir cosas igual de interesantes. Y todos nos poníamos *bizcos de placer* pensando lo que se podría hacer con semejante conocimiento... o mejor, lo que yo, o mi empresa, podríamos hacer con semejante conocimiento. No obstante ser falso, voy a utilizar el conocido y sencillo de entender ejemplo para intentar explicar qué es eso del Data Mining, hablando a mi rústica manera sobre los conceptos más importantes que hay ahí enterrados.

Vamos a ellos.

En primer lugar, **sobre la propia definición de la disciplina del Data Mining**. Muchas definiciones similares, pero distintas, podréis encontrar sobre qué es eso de la Minería de Datos. A mí la que más me gusta de las que han caído en mis manos (ignoro la fuente, sinceramente) es: **“Data Mining es el Proceso de descubrimiento mediante métodos automáticos, sobre las Bases de Datos, de Conocimiento valioso y no evidente que se encuentra enterrado en la semántica intrínseca de los datos”**.

En nuestro manido ejemplo queda claro que el Conocimiento obtenido es *No Evidente*, pues no creo que nadie en sus cabales imaginara siquiera tal relación entre cerveza y pañales; en cuanto a lo de *Valioso*... bueno, le concedemos el beneficio de la duda. Lo que es también claro es que es un descubrimiento que ha debido de hacerse de forma automática, o sea, un programa informático con infinita paciencia (o, mejor, ejecutando un buen algoritmo de descubrimiento) ha ido probando combinaciones improbables hasta encontrar una que obtiene un resultado interesante... no me imagino yo a un sesudo analista probando posibles combinaciones una tras una: ¿Leche y Galletas? ¿Lejía y Comida kosher? ¿Vino blanco y Pimienta? Las combinaciones son innumerables hasta llegar a ¿*Pañales y Cerveza*?

En segundo lugar, **sobre la utilidad en sí del descubrimiento**. No tengo ni la menor idea de qué podría hacer WalMart para convertir un conocimiento tan *privilegiado* en una mejora sustancial de su negocio, es decir, conseguir más venta o más margen, que es como mejoran su negocio las grandes compañías comerciales.

Quizás hacer una promoción “*Compre una docena de pañales y le regalamos una birra*” o, mejor aún: “*Comprando dos docenas de cervezas le regalamos dos pañales King-Size, que si su hijo aún no los gasta, ya los gastará*”. O poniendo juntos los Pañales con las Cervezas en la misma sección del supermercado, para que los pocos desmemoriados que vayan a comprar una cosa no se olviden de llevarse también la otra. Para mí que, de ser cierta, esta relación Pañales-Cerveza no sería muy útil para aumentar las ventas, pero...

doctores tiene la Iglesia.

Ojo, que aunque en este ejemplo se ve malamente qué utilidad inmediata podría tener para mejorar la cuenta de resultados, puede haber otros casos, y los hay, lo aseguro, en que un descubrimiento de este estilo sí que puede tener una enorme utilidad, bien aumentando ventas, bien reduciendo costes o riesgos, etc.

En tercer lugar, **sobre la inferencia de por qué se produce ese hecho específico de negocio**. Resultaba que está claro, clarísimo, *cristalino*, que eran hombres, y jóvenes, además, los que compraban la cerveza para ponerse ciegos mientras ven el partido de los Bulls contra los Sixers, mientras el niño pequeño berrea y su madre le cuida y le cambia los pañales, ¿no? Pues no. Eso es un estereotipo que no tiene por qué ser verdad... A ver por qué extraña razón eso tiene que ser así y no es de cualquier otra manera.

La realidad pura y dura es que el descubrimiento, si hubiera sido real, que ya sabéis que no lo fue, nos dice que hay clientes que compran simultáneamente pañales y cervezas los sábados por la tarde... **y punto**. Otras interpretaciones sin datos que las avalen pueden conducir mayormente a monumentales errores en las decisiones tomadas.

Y en cuarto y último lugar, **sobre el propio hecho de que un “importante y trascendental” descubrimiento para WalMart fuera tan conocido en todo el resto del mundo mundial...** Si yo fuera WalMart y conociera algo realmente interesante sobre el comportamiento de mis clientes que puedo aprovechar para, inmediatamente, generar mayores ventas o beneficios donde, además, cualquier estrategia que ponga en marcha para explotar lo descubierto es fácilmente replicable por mis competidores... **lo último que haría sería comunicarlo a los cuatro vientos**. Me callaría como un muerto y explotaría el filón todo lo posible, esperando que los buitres de mis competidores no se enteren o, al menos, que lo hagan lo más tarde posible, ni de la existencia en sí del nicho, ni de cuáles son mis acciones para explotarlo.

Creo que, tras desmenuzar un poco lo que hay detrás de una adquisición de conocimiento de este tipo, quizá haya quedado más claro qué es y para qué sirve esa cosa del Data Mining. Nombre que, por cierto, a mí me parece espantoso. A mí, *Minería de Datos* me evoca un trabajo penoso, sucio, costoso, con cierto riesgo, para arrancar a la montaña de datos los escasos diamantes de gran valor que están ocultos en su interior...

Será una descripción muy adecuada de lo que es el trabajo de Descubrimiento. Pero **no es muy comercial**, puesto que el principal objetivo de la Minería de Datos no es el propio proceso de cavar y cavar, sino entender realmente el **Significado** oculto de los datos.

Yo tengo una *teoría estúpida* de las razones últimas de ese nombrecito... Ahí va (no ibais a esperar que, a estas alturas, me la dejara en el tintero, ¿no?).

Alguien, inglés o estadounidense, por supuesto, en cualquier caso, angloparlante, definió esta actividad en sus comienzos y la bautizó con el nombre más lógico y adecuado, a saber: “**Data Meaning**”, dado que lo que queremos es conocer el **significado de la información**... Se lo contó a algún técnico o ejecutivo español y el hombre, con sus proverbiales escasos conocimientos de anglosajón, apuntó aplicadamente en sus notas “Data Mining”. Total, como suena igual... ¡A quién se le ocurriría que en inglés la *i* se pronuncie *ai* y que el diptongo *ea* se pronuncie *i*... a veces!

Al llegar a España apuntó el término en alguna presentación, la presentación cayó finalmente en manos de algún genio del marketing, la cosa hizo gracia... y la disciplina fue finalmente rebautizada.

Fin de mi *estúpida teoría*. Posible, ¿no?

Fueron los Departamentos de Marketing de todo tipo de empresas los que empezaron a demandar un tipo de conocimiento que no estaba disponible. Si os acordáis de mis cuitas con el responsable de aquel banco hace treinta años o más, cuitas que os conté en el capítulo dedicado a los servidores de proceso paralelo, los Departamentos de Marketing tenían muy poco soporte de Aplicaciones Informáticas... y poco que hacer, además. Tomaban información de la Base de Datos de Clientes con los criterios que buenamente se les ocurrían, pero no eran capaces ni tan siquiera de poder averiguar el alcance ni menos aún el resultado de sus campañas publicitarias... Me explico:

Imaginemos un gran Banco que hace una campaña de publicidad para ofrecer un nuevo producto, digamos una Tarjeta de Crédito cuando en España todo el mundo pagaba en pesetas contantes y sonantes. Para ello, utiliza técnicas de *marketing mix*, es decir, varios medios (canales, en la jerga) para publicitar el nuevo y maravilloso producto. Anuncios en Televisión, Anuncios a página completa, o Insertos en Prensa, Cuñas en Radio, Vallas Publicitarias... y además hace un envío de folletos a los domicilios de sus clientes, a todos ellos, o sólo a unos pocos, los que a priori se piensa que van a ser más proclives a tan novedoso producto financiero. Y, naturalmente, se instruye a la fuerza comercial de la red de oficinas para que ofrezca y venda el producto a todo cliente que pise una oficina... me imagino que sabéis de qué estoy hablando. Se conceden (o pagan, que también) entrevistas a prestigiosos periodistas, entrevistas que se publican en los diarios más influyentes... un gran despliegue, vaya.

La campaña sigue su curso y, al cabo de dos meses o tres, cuando se decida, se hace balance. Dos cosas son, básicamente, las que pueden haber ocurrido:

**Caso 1: La campaña fue todo un éxito.** Nos hemos hartado de vender tarjetas. Todo el mundo se congratula: el Producto Diseñado es magnífico y el momento elegido para su lanzamiento, perfecto; la Campaña publicitaria ha sido formidable, incluyendo la elección de la actriz de moda apropiada para convencernos de lo útil del invento; el comportamiento de la fuerza comercial ha sido extraordinario, consiguiendo altísimas cotas de aceptación del producto; el folleto estaba bien diseñado y se ha enviado con precisión de cirujano exactamente a los clientes más interesados.

Vamos, que el éxito ha sido gracias a todos y cada uno de los intervinientes... en exclusiva. Y como en definitiva la cosa ha ido bien, pues nada, felicitaciones, parabienes y subidas de sueldo para todos, y listo...

... Aunque la realidad pura y dura es que **nadie sabe las causas reales por las que la campaña haya resultado un éxito**, nadie tiene la más remota idea, por lo que es muy difícil capitalizar el conocimiento necesario para poder repetirlo en un futuro con otro producto diferente.

**Caso 2: La campaña resultó un fracaso.** No se llega ni de lejos a los objetivos marcados. Entonces, la imagen más parecida que me viene a las mientes para representar la situación es la de las ratas huyendo por las troneras antes de que el barco termine de naufragar...



Éste es uno de esos momentos en que todo el mundo asegura aquello tan español de “*Lo mío está bien*” y culpa al de al lado, a *todos los de al lado*, en realidad, del fracaso de la campaña.

Es decir: la campaña ha fracasado porque el producto no estaba bien diseñado, porque se ha lanzado en un momento poco idóneo, los anuncios de televisión eran largos, o quizá cortos, eran chabacanos o quizás demasiado sofisticados, o tal vez el actor estaba mal elegido, con su carita de niño bueno, los comerciales en las oficinas han pasado del tema... y así con todo.

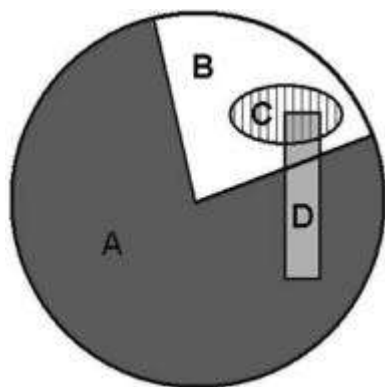
La realidad pura y dura es que las campañas a veces son un éxito y a veces fracasan, no hay quien tenga siempre la bola de cristal en perfecto estado de funcionamiento... **lo realmente importante es averiguar las razones del éxito o del fracaso** para repetir las en el futuro, las primeras, o evitarlas, las segundas, no convertir la campaña en una carrera a ver quién ha hecho mejor su trabajo, en el primer caso, o quién *no* tiene la culpa, en el segundo.

Por aquellos finiseculares años se comenzó a poner de moda un nuevo concepto: el *micromarketing*. Es decir: dirigir a cada cliente las ofertas de productos en que esté interesado, evitando costosas campañas masivas de escasa penetración y alto coste. No tiene sentido intentar colocar un Fondo de Inversión a un cliente que no nos paga la Hipoteca. Aseguro que yo he visto casos como ése, o peores...

Pero no se trata sólo de cosas tan sencillas como esa tontería, porque para eso no hace falta Data Mining de ningún tipo, tan sólo tener *un poco de sentido común*. Lo interesante de usar técnicas de Minería de Datos aplicadas al micromarketing es que permiten localizar nichos interesantes y *completamente desconocidos previamente*.

Pongamos un ejemplo sencillo: Vamos a lanzar un Fondo de Inversión, por ejemplo, y se lo ofrecemos selectivamente a un grupo de clientes que a priori nos parece que pueden estar interesados, sea como fuere que hayamos adquirido este convencimiento.

En el círculo de la imagen vemos un esquema de cómo sería esa campaña.



Dentro del círculo tenemos a todos los clientes del Banco susceptibles de recibir la oferta (hemos eliminado los que nos deben dinero y cosas así). De ellos hemos seleccionado más o menos a la quinta parte, el segmento circular con forma de quesito de la derecha, y a ellos, y sólo a ellos, nos hemos dirigido para ofertarles nuestro magnífico producto personalmente.

Así que a los clientes del grupo A (el grupo gris oscuro) no les hemos ofrecido nada, por lo que nada pueden comprarnos, mientras que al grupo B (el grupo blanco) sí que nos hemos dirigido por carta, llamándoles por teléfono, por email o como sea.

Tras el final de la campaña miramos a los que sí nos han comprado el producto, es decir, donde hemos tenido éxito, y resulta que es el grupito C (el rayado). Supongamos que ese grupo representa el 6% de todos los clientes del grupo B, los receptores. Ese 6% puede ser bueno o malo, depende, pero no se trata de eso, sino de aprender para conocer a nuestros clientes...

Empezamos, pues, a estudiar qué ha pasado en la campaña. Y tras mucho cavilar, descubrimos que hay un grupo de clientes D (*el grupo de color gris más claro*) que son, por ejemplo, aquellos que tienen una Hipoteca de Segunda Vivienda, a algunos de los cuales les incluimos en su día en la campaña y a otros no.

Y vemos que de todos los clientes de ese grupo D a los que sí les hemos ofrecido el Fondo, es decir, la intersección del grupo D con el grupo B, resulta que hay un 42% de clientes que sí lo han contratado: el grupito marcado en gris claro y rayado, que es la intersección de los grupos B, C y D, mientras que el 58% restante no lo han hecho.

Como sólo el 6% de todos los clientes a los que se lo hemos ofrecido (los del Grupo B) lo han contratado, parece que ese nicho concreto, *Clientes con Hipoteca de Segunda Vivienda*, son mucho más proclives a interesarse en nuestra campaña, exactamente siete veces más (el 42% contra el 6%), que nuestro *cliente medio*. Si lo hubiésemos sabido en el momento de diseñar la campaña, está claro que hubiéramos hecho la oferta a todos los clientes de ese grupo tan particular, sin dejar fuera ni uno... pero no lo hicimos porque no lo sabíamos.

Lo importante no es mesarse los cabellos por la oportunidad perdida, eso es una pérdida de tiempo. **Lo importante es aprender cómo son nuestros clientes** para que, cuando tengamos que hacer un nuevo ofrecimiento de otro producto de inversión, como un depósito o un nuevo fondo, seleccionemos en primer lugar a todos esos clientes del Grupo D, sin desechar a ninguno, dado que parecen muy interesados en estas ofertas, mejor dicho, *nos han demostrado* que son proclives a ella de la única forma que no admite discusión: contratando el producto.

Por cierto, esto que estoy haciendo con los diferentes Grupos de Clientes se llama

**Segmentar** y existen varias técnicas entre todas las de Data Mining que están orientadas a obtener segmentos de datos: de clientes, de productos, o de lo que sea.

Todas las empresas comerciales se lanzaron, a mediados y finales de los noventa del siglo pasado, a explorar estas nuevas posibilidades con mejor o peor éxito. Pero hubo un sector concreto de empresas que lo hicieron con muchísimo más ahínco que ninguno: las “Telcos”, nombre genérico que reciben las grandes empresas de telefonía y comunicaciones, debido no sólo a su potencia financiera, sino al hecho de que esos años se estaba produciendo una brutal desregulación en el mercado de las telecomunicaciones en todo el mundo, abriendo los tradicionales monopolios nacionales a la competencia... Telecom Italia, Deutsche Telekom, France Telecom, British Telecom... y en España, Telefónica, claro.

Todas ellas tuvieron, de grado o por fuerza, que compartir sus hasta entonces monopolizados mercados con nuevas empresas que irrumpieron cual elefante en cacharrería con fortísimas inversiones que había que amortizar. Y las que ya estaban procuraban conservar a toda costa sus clientes... perdón, sus *abonados*, que fue más o menos entonces cuando los abonados nos fuimos convirtiendo en clientes... ¡Qué buenos tiempos aquellos, donde llamabas y te atendía una señorita (a veces un caballero, pero pocas) que te solucionaba tus problemas!, en vez de tener que tener que entenderte con una máquina parlante que está entrenada para marearte, aburrirte y que cuelgues, eso sí, habiendo llamado a un costoso número 902... En fin.

No sólo telefónicas varias entraron en el mundo de la segmentación y el micromarketing. Toda empresa que se preciara de todos los sectores entraron también para mejorar su Sistema Comercial. Bancos, Aseguradoras, Distribuidoras, Fabricantes... Fue cuando comenzamos a oír hablar de los sistemas de CRM, Database Marketing...

Y empezaron a llegar noticias de éxitos, debidamente aireados por fabricantes y consultores... Por ejemplo, una Compañía de Seguros británica encontró un nicho de clientes excepcionalmente rentable: una vez estudiado, resulta que se trataba de *coleccionistas de coches clásicos* (se ve que en el Reino Unido había bastantes de estos).



Se trata de clientes que apenas circulan con sus preciados juguetes, como el Triumph de la década de los 50 de la foto, que los cuidan y miman más que Sebastian Vettel a su Formula 1... y sin embargo los tienen todos asegurados. Pagan religiosamente sus pólizas, pero, como no mueven sus coches, no tienen siniestros. Un chollo para la Compañía de Seguros.

Esta compañía sí que encontró cómo rentabilizar su descubrimiento: preparó una póliza especial para coleccionistas con condiciones que eran sensiblemente mejores que las habituales y utilizó a los *Royal Automobile Clubs* de todo el país para ofrecérsela a todos los coleccionistas británicos... El objetivo era robárselos a la competencia incrementando, por tanto, su negocio y sus márgenes... una temporada solamente, claro, puesto que el resto de competidores reaccionaron al cabo de poco tiempo, ofreciendo pólizas similares... pero *quien da primero, da dos veces*.

Buena parte de esas *ofertas extrañas* de las telefónicas, de bancos o de todo tipo de

compañías de todo pelaje que amablemente nos hacen día sí, día también, se basan en el descubrimiento de nichos de mercado, que procuran aprovechar antes que la competencia los descubra.

Todos los CRM's permitían controlar las diferentes campañas publicitarias por fases, con la selección de los clientes más propicios... sólo que esto último es, en buena medida, mentira. Porque eso de seleccionar los clientes propicios no lo puede hacer *automáticamente* una herramienta solita sin la colaboración de un humano que se conozca bien todo el percal: lo más importante para poder hacer buenos estudios de Data Mining siguen siendo los paisanos... Volveré a esto más adelante.

Las técnicas en que se basa la Minería de Datos son, con alguna excepción, conocidas desde hace mucho tiempo y se basan tanto en la Estadística como en las Ciencias de la Computación y en la Inteligencia Artificial. Sin embargo, sólo el advenimiento de potentes máquinas de proceso en paralelo permitió de verdad realizar estudios sobre una cantidad considerable de datos... Así, a bote pronto, podemos citar entre sus técnicas más utilizadas el Análisis Factorial, la Regresión Lineal o Logística, Asociación, Segmentación (Clustering), Análisis Chaid, Predicción (Forecasting)... y varias más.

No, no voy a describir ni detallar ninguna de estas técnicas, podéis respirar tranquilos; siendo como es uno de los *hot-topics* del momento, hay en la red toda la información que deseéis, mucho mejor estructurada y detallada de lo que este pobre informático vejastorio y lenguaraz pudiera hacer, así que... ésa es otra historia y será contada en otro momento, o mejor, *por otro*.

Lo que sí cambió a finales de los noventa fue **la posibilidad real de poder aplicar cualquiera de estas técnicas a grandes volúmenes de información**, gracias a los nuevos servidores de gran potencia de cálculo y capacidad de almacenamiento, así como nuevas implementaciones de algoritmos conocidos desde hacía décadas para aprovechar el paralelismo ofrecido por la tecnología. Si a todo eso le añadimos una mucho mayor capacidad de visualización de los resultados, tanto de forma tabular como gráfica, y el que los datos origen para hacer el estudio, habitualmente contenidos en el Data Warehouse, tenían un formato mucho más normalizado y unificado que cuando residían en sus Bases de Datos originales, el Data Mining se hizo, por fin, posible. *Más o menos posible*, al menos.

Dos productos existían ya a finales de los noventa con fantásticas capacidades estadísticas, pues ambos nacieron como programas especializados en la resolución de problemas de ese cariz, y ellos fueron de los primeros que se apuntaron al carro del Data Mining expandiendo sus ya bien estructuradas suites para cubrir aquellas áreas o técnicas que no lo estaban.

Uno de estos productos fue **SAS**, de excelente calidad y poderosos módulos estadísticos, que pronto unificó inteligentemente todos los módulos necesarios en el *SAS Enterprise Miner*, producto caro y complicado de usar pero potentísimo para tratar con volúmenes muy elevados de datos. El otro fue **SPSS**, muy bien posicionado en ambientes académicos y en sectores clave, fundamentalmente el sanitario.

Me contaron hace ya algunos años que el Institut Català de la Salut, encargado esos años de la Sanidad Pública en Cataluña, se enfrentó a un problema sanitario realmente curioso... y preocupante: la incidencia de determinados tipos de alergia en la ciudad de Barcelona tenía unos picos de incidencia elevadísimos en ciertos momentos del año. Esos momentos no respondían a ningún patrón conocido como la germinación de la arizónica o de las gramíneas, y se repartían de forma irregular a lo largo del año. Además, estos brotes

eran muy virulentos y afectaban a muchas personas simultáneamente, que debían recibir atención médica y ser, en muchos casos, dadas de baja por enfermedad. Aparentemente, sólo una causa externa podía justificar esos cuadros alérgicos, pero... ¿cuál?

Varios investigadores se pusieron manos a la obra, incidentalmente, usando SPSS, hasta que varios meses después un genio encontró por fin una correlación casi perfecta entre las **descargas de soja** en el puerto de Barcelona y, un par de días después, la aparición de los brotes.

Los buques de transporte de grano lo llevan a granel en sus bodegas y para descargarlo se empleaba un aspirador, o quizás un elevador de granos que recogía el grano de la bodega y lo llevaba hasta el silo. Toda la operación se hacía al aire libre y generaba una enorme cantidad de polvo... que el viento repartía por toda la ciudad y ponía muy malitos a muchos pobres alérgicos.



Una vez conocida la causa fue sencillo poner remedio: cerrar herméticamente los silos y cuidar el procedimiento de descarga para evitar que se produjera polvo. Fácil, ¿no? Lo que le extraña a este viejo informático es que, sabiendo como hoy se sabe esta relación entre polvo y alergia, siga habiendo puertos en los que hoy en día se descarguen los granos de la manera que se puede observar en la foto... ¡Obsérvese la polvareda! Está claro que los humanos no aprenderemos nunca...

Al menos, espero que los barcos de transporte de grano sólo se dediquen a transportar *grano* de esta forma, porque como entre carga de soja y carga de cebada les contraten para hacer un transporte de cemento... Bueno, en realidad creo que es mejor no saberlo.

Muchos productos de Data Mining aparecieron al calor de la novedad; unos tenían más funciones, otros menos... unos eran mejores y otros peores, todos competían por el emergente mercado naciente y unos tuvieron más éxito que otros... lo de siempre, vaya.

IBM concentró una serie dispersa de algoritmos que había ido desarrollando a lo largo de los años en su Suite **Intelligent Miner**; hizo alguna venta pero no demasiadas, pues apenas tenía técnicos en España para implantar la solución, y ahora ha sido parcialmente incorporado dentro de la propia Base de Datos DB2. Y en julio de 2009, IBM ha adquirido a otro de los grandes jugadores del partido: SPSS, cuyos productos habrán sido rebautizados e integrados en vaya Vd. a saber qué suite.

**Angoss**, **Clementine** y otras varias más también tuvieron su momento de gloria... luego llegó la consabida “consolidación del mercado” y la mayoría fueron compradas (como Clementine, comprada por SPSS, por ejemplo, y luego ésta por IBM) o simplemente desaparecieron. SAS y SPSS/IBM son, hoy en día, los vencedores de la guerra del Data Mining... con alguna excepción proveniente del software libre: **Weka**, creado y mantenido



por la universidad de Waikato, Nueva Zelanda, tiene una potencia mediana; es una excelente herramienta libre muy recomendable para comenzar a jugar con el Data Mining.

Pero... **la clave son las personas**. Siempre las personas. Por mucho producto *atómico* que compremos, hay que saber utilizarlo... y eso no es nada, pero nada fácil de encontrar. El perfil adecuado para que alguien haga buenos estudios de Data Mining incluye un excelente conocimiento del negocio para seleccionar e interpretar la información obtenida. Obviamente, debe conocer bien las diversas técnicas utilizadas, para qué sirve cada una y cuándo utilizar una u otra, o bien confirmar los hallazgos realizados con una mediante otra diferente... Además, tiene que ser capaz de convertir y manipular datos para adecuarlos al estudio concreto a realizar, categorizando variables continuas, eliminando información irrelevante, etc.

Resumiendo, se trata de un perfil difícilísimo de encontrar en cualquier empresa, por grande que sea... pero es que tampoco es nada sencillo encontrar un consultor que cumpla los requisitos.

Y el caso es que, sobre todo a las empresas financieras, no les queda más remedio que encontrar ese perfil, debido ni más ni menos que a la existencia de una cosa llamada *Acuerdos de Capital de Basilea (II, III y quién sabe en qué número acabarán)*.

Ah, claro, que muchos de los lectores nunca ha oído nada de la preciosa ciudad suiza de Basilea, salvo como destino turístico, o quizá sí hayáis oído hablar de “*Basilea II o III*” pero no sabéis qué es eso ni para qué sirve. Pues nada, nada, aquí estoy yo para explicarlo, si es que os interesa...

En Basilea se encuentra el *Banco Internacional de Pagos*, donde se cocinan muchas de las regulaciones y normas que afectan a la Banca Internacional.

En el año 1988 se firmó el **Acuerdo de Capital de Basilea**, el primero, que entre otras cosas obligaba a los Bancos a realizar provisiones para garantizar sus riesgos, es decir, la posibilidad de que sus créditos y préstamos (que no son lo mismo) resulten impagados. Esas provisiones consisten en que tienen que depositar un cierto porcentaje de la cantidad prestada en el Banco Regulador (en España, el Banco de España), sin interés ninguno o con muy bajo interés. El porcentaje que representa estas provisiones tiene que ver con el tipo de riesgo; si mi memoria no me engaña, para el crédito hipotecario es el 2,25%; para riesgo no hipotecario a empresas y particulares, el 5% y para crédito al consumo, sobre todo las tarjetas de crédito, nada menos que el 8%.

Esas provisiones aseguran que, en caso de que una parte de los créditos concedidos por ese Banco resulten morosos, no afecte a la solvencia del Banco y no se lleve los dineros de sus depositantes, aunque de todos modos ya hemos visto que eso tampoco pasa: si un Banco tiene problemas, papá Estado, o sea, nosotros, los contribuyentes, acudimos a salvarlo. *Basilea I* sirvió para mejorar la solvencia de la Banca Mundial, pero ya a principios de los dosmiles estaba claro que, por un lado, estaban apareciendo nuevos instrumentos financieros “*sofisticados*” (léase “pensados para limpiarte la plata sin que te des ni cuenta”, que incluso yo he picado en un par de esos) que, además, se escapaban por los flecos que Basilea I dejaba abiertos.

Por si fuera poco, en un ciclo expansivo de la economía tras el fiasco de la “*burbuja puntocom*” de la que hablaremos más adelante, esos porcentajes se antojaban altísimos... para la propia banca, claro está. ¡*Hombre, por Dios!* Con lo bien que se selecciona el riesgo hipotecario en este Banco mío, un 2,25% de provisiones es altísimo, a quién se le ocurre... Sí, sí... *No comment*.

La solución: **Los Acuerdos de Capital de Basilea II**, firmados en 2004, que

constan de un montón de regulaciones y recomendaciones, pero que, mayormente, consiste en que los Bancos establezcan “*sofisticados*” sistemas de control del riesgo, basados en técnicas estadísticas y de tendencia de extraordinaria solvencia y exactitud, sistemas que ellos mismos definen... no, esperad, que es peor aún: cada banco define su propio sistema estadístico “*sofisticado*” de valoración del riesgo, lo pone en marcha y lo usa.

Básicamente mide dos parámetros: la probabilidad de “*default*” (impago) de un cierto crédito a lo largo del tiempo y el monto esperado de parné que, en caso de *default*, va a dejar a deber el prestatario al banco. Y es fundamental que todos estos sistemas funcionen bien, pues la Gestión de Riesgos es, en realidad, como un gran castillo de naipes: si se retira según qué naipe, se derrumba todo el chiringuito. ¡Qué os voy a contar de bancos malos, rescates y demás martingalas!

Los Bancos reguladores (en España, el Banco de España) revisan que realmente el Banco ha implantado un “*sofisticado*” sistema basado en la estadística y que lo cumple a rajatabla... y entonces, las provisiones que debe realizar el Banco o Caja de Ahorros se reducen sustancialmente... hasta más de la mitad, según los casos. Y eso representa muchísimos millones. No de pesetas, no... de euros, de dólares... Y mientras tanto, mientras no tengan implantados esos “*sofisticados*” Sistemas de Control Crediticio... siguen funcionando las elevadas provisiones antiguas... sí, ésas que con el tiempo se han demostrado claramente insuficientes... así se escribe la historia.

Por lo tanto, todos los Bancos y Entidades Financieras se adhirieron entusiásticamente a Basilea II, lo que les obligó a entrar de lleno, mal que les pesara, en el proceloso mundo de la estadística o del Data Mining, dando el espaldarazo definitivo a este tipo de productos.

SAS, por ejemplo, ha vendido cientos de licencias de su SAS Enterprise Manager en todo el mundo bajo el paraguas de Basilea II, los consultores han vendido miles y miles de horas... Mucho negocio para mucha gente. Así que a los pocos años se inventó “Basilea III”, y seguro que luego inventarán “Basilea IV”, “Basilea LXXVII”, etc.

En fin. Buena cosa, eso de Basilea II o el que toque, ¿no?

Pues no me preguntéis qué opino personalmente de Basilea II y que sean los propios bancos los que se “*autorregulen*” para conceder riesgos; no hay más que ver cómo han acabado la mayoría de “*autorregulados*” en los últimos meses para constatar la eficacia de la “*autorregulación*” financiera, pero ésa es otra historia y será contada en otra ocasión.

La cruda realidad es que, antes de Basilea II, cualquier inspector del Banco de España, en un día y con una simple hoja electrónica, era capaz de auditar todos los parámetros básicos de una entidad financiera. Ahora, ni en un mes, y además necesita conocimientos estadísticos y de Data Mining al alcance de muy pocos: son inspectores financieros, no estadísticos. Esta historia va a acabar muy, pero que muy mal... De hecho ya *está* acabando mal, qué os voy a contar.

En fin, nominalmente muchas empresas tienen Sistemas y Departamentos dedicados a realizar estudios de Data Mining. A los Bancos, si quieren tener las ventajas que Basilea II les ofrece, no les queda más remedio que tener uno asociado al departamento de Riesgos. Si no, *no cuela*, y no pueden beneficiarse de las ventajas que les ofrece el nuevo Acuerdo. Nuevo acuerdo que ya es viejo, porque está ya en la palestra Basilea III, con quién sabe qué nuevas regulaciones y normativas...

Muchísimas empresas comerciales, que dedican grandes cantidades de dinero al marketing, también hacen estudios más o menos sistemáticos, aunque lo más normal es que

estas funciones estén embebidas en los Sistemas CRM que tengan instalados.

Ignoro su uso real... pero es que estoy convencido de que las empresas que lo usen de verdad y obtengan resultados interesantes para su negocio se los callarán como un muerto: yo me los callaría. Si tengo una empresa dedicada a la venta de zapatos y descubro que los clientes que tienen más de dos pies compran el doble de zapatos al año que los que tienen dos pies o menos, haría una campaña para explotar el nicho descubierto y sería como una tumba, para que la competencia no se entere y me pise, nunca mejor dicho, el nicho.



...Y así hasta que el dichoso nicho esté completamente explotado y haya extraído todos los diamantes que contiene, como en la famosa mina de diamantes de Kimberley, en Sudáfrica, que, explotada durante cincuenta años (desde 1866 hasta 1914), de ella sólo queda un agujero inmenso lleno de agua... y sin un solo diamante dentro, según se ve en la imagen, cortesía de la Wikipedia.

La información que hay en el mercado sobre éxitos en la aplicación del Data Mining es escasa y, sobre todo, poco fiable. No me creería yo mucho esos “casos de éxito” que nos presentan los consultores para demostrarnos lo bien que hacen las cosas.

**La clave, repito, son las personas.** Algunos consultores tienen gente muy preparada, qué duda cabe. Pero **hay que traspasar el conocimiento a las personas de la empresa.** Y desgraciadamente, al menos en España, los destinatarios naturales de dicho conocimiento son poco receptivos.

Hay poca formación básica en los responsables de estudios, analistas de negocio, etc., y menos aún en los Directores, para la mayoría de los cuales una *Desviación Típica* es... ¡la homosexualidad! Y no, no os riáis, que ésta es una anécdota real que yo he oído preguntar a un *Director General de Muchas Cosas* de una importante compañía manufacturera española.

Se manejan muchísimos datos, ciertamente, pero el único estadístico que la gente entiende (más o menos) es la media, a veces, sin ponderar siquiera. Por ejemplo: “*el consumo medio de los turistas extranjeros en España es de 94 Euros por día*”, fue una noticia que saltó a toda la prensa en mayo de 2009. ¿Y ese dato tan sesudo para qué vale? ¡Al menos habría que acompañarlo con la desviación típica!

Por ejemplo: “una media de 94 Euros por día con una *DT* de 78 Euros”, refleja una realidad completamente distinta de si es “94 Euros de media con una *DT* de 5 Euros”. Con la Desviación Típica me hago una idea, quizás parcial y con no mucho valor añadido, pues sería mejor usar más estadísticos, los que sean más descriptivos en cada caso, por ejemplo percentiles, segmentado en los principales grupos... pero ¿sólo la media? ¿Qué información da eso, para qué podemos usarla? ¡Para nada! En fin...

...Pero, claro, es que **nadie entiende ningún estadístico.** Yo lo sé: he hecho informes donde he utilizado cuatro estadísticos tontorrones para intentar explicar lo que pasa en algún sitio (mis oxidados conocimientos de Estadística de Tercero de Carrera tampoco dan para más) y nadie entiende otra cosa más que la media, la dichosa media, que

a veces está, además, mal calculada, para más *inri*.

Pocos saben interpretar un valor de percentil una moda o una mediana, qué significa, para qué sirve... eso si no te dicen directamente que “*eso de la estadística es una engañaifa...*”, que esto también lo he oído yo con mis propios oídos, y descartan por irrelevante e inicuo todo dato que contenga un estadístico... que sea *diferente de la media*, claro, la omnipresente y ubicua media.

Tenéis mucho por hacer, jóvenes lectores, si queremos cambiar esto alguna vez. En vuestras manos está; yo no he podido o sabido, y ¡*por Tutatis!* que lo he intentado, pero se ve que los tiempos no habían llegado.

**¡Suerte!** Aunque... en fin, no sé por qué me da que lo vais a tener aún peor que yo lo tuve.

## 21 - La “*burbuja puntocom*” se hinchó... y explotó

Revoloteando siempre alrededor de la historia del Data Warehouse, pero también de todas las demás historias que ocurrían simultáneamente en la tecnología informática en la segunda mitad de los noventa (y había muchas), estaba el espectacular ascenso de Internet durante todos esos años noventa del Siglo pasado.

Inicialmente era una cosa *Top-Secret* restringida al uso militar, pues es bien conocido que el origen de Internet está en el diseño de Arpanet, una red cuya principal función era garantizar el acceso a la información por parte de los principales centros de mando estadounidenses en caso de catástrofe... esas cosas tan lindas y divertidas que se tenían permanentemente en cuenta durante la guerra fría: había que ser capaces de ganar a toda costa la hipotética guerra, aunque no quedara nada que gobernar después... Sin embargo, durante la década 90 se había ido convirtiendo con los años en algo de acceso común, en un artículo más de consumo que llegaba, o sería capaz de llegar en pocos años, a la práctica totalidad de la población mundial.

Y eso, en definitiva, significa negocio. **Mucho negocio** que un buen visionario no puede permitirse el lujo de ignorar. Así que multitud de empresas de tecnología tradicionales, nuevas iniciativas de pequeñas empresas y miles de activos emprendedores volvieron sus ojos hacia el nuevo Eldorado, con sus promesas de oro ilimitado para el listo que fuera capaz de encontrarlo... Y la burbuja comenzó a hincharse...



...Y conoceréis sin duda que sólo unos pocos años después la dichosa burbuja explotó con gran estrépito, salpicándonos bien a todo el mundo con sus detritus. Ya se va cansando uno de la cantidad de burbujas que le han explotado en la cara sin comerlo ni beberlo...

De mis recuerdos y sensaciones durante este periodo trata este capítulo; lo que dio en llamarse “burbuja puntocom” (*Dotcom bubble*, en inglés) es un tema muy conocido, está muy documentado y, como además ocurrió hace relativamente pocos años y está aún en el recuerdo de la mayoría, no esperéis ninguna crónica oficial de nada, sólo mis recuerdos, los fragmentarios e indocumentados recuerdos de un Viejo y Asombrado Informático que pasaba por allí...

Como acabo de decir, el tatarabuelo de Internet es Arpanet. Diseñado por encargo del Gobierno de los Estados Unidos, en concreto por la *Agencia de Proyectos Avanzados de Investigación* (ARPA) como un método de comunicación directo entre ordenadores, estaba pensado para no dejar sin mando a las Bases y Centros de Mando del Ejército

Norteamericano en caso de que el host, el ordenador central, dejara de funcionar. No porque se fundiera un fusible, no, que eso pasaba cada lunes y cada martes... sino porque sufriera un ataque nuclear del archienemigo del momento, de la URSS.

La Wikipedia niega todo esto, lo de que uno de los objetivos de diseño fuera sobrevivir a un ataque nuclear, pero lo he oído tantas veces y, sobre todo, me parece *tan lógico* desde la paranoica visión de la milicia de entonces, que me extrañaría que no fuera, al menos en parte, cierto.

Arpanet fue una Red de Comunicación de Paquetes como lo es ahora Internet y, tras varios años de definiciones, las especificaciones estuvieron por fin listas en 1968; en 1969 se asignó el proyecto a una empresa tecnológica de nombre BBN Technologies y, como los norteamericanos trabajan bien y rápido, a fines de 1969 estaban ya conectados los principales nodos. ¡Los estadounidenses habían inventado y puesto en marcha la conmutación de paquetes!, tecnología en la que se basa ampliamente la tecnología de internet de hoy en día. Eran unos genios...

Pero... **¡un momento!** Todos los proyectos de ARPA y entre ellos, cómo no, Arpanet, eran secretos. Muy, pero que muy supersecretos...

Pues ya veis, a pesar de eso, resulta que en España, en la CTNE, la “Compañía Telefónica Nacional de España” de antaño, o sea, la Telefónica o la Movistar de hogaño, a finales de los sesenta, cuatro ingenieros iluminados (y no exagero, creo que fueron cuatro o cinco, no más) **habían definido exactamente lo mismo** que los mejores ingenieros norteamericanos del MIT y demás... en mucho menos tiempo y por cuatro perras: el presupuesto para poner la RETD en España en marcha, con seis nodos operativos, era de menos de 200 millones de pesetas de la época: ¡1,2 millones de euros! No 1.200 millones, no: *¡uno coma dos millones!*

Por mucha inflación que haya habido en España desde entonces, que sí que la ha habido, ya me gustaría saber cuántos cientos de millones de dólares de los de entonces se gastó el Gobierno americano en poner el supersecreto y superpoderoso Arpanet en servicio...

Esa maravilla fue bautizada como **RETD** (Red Especial de Transmisión de Datos, qué excelente invento... y qué nombrecito tan poco comercial), funcionó perfectamente desde 1971 y dio un magnífico servicio... a las empresas, claro, no a los raquíuticos centros militares españoles de la época. Jesús Martín Tardío lo contó en un fantástico relato, que ya cité en otro capítulo anterior y que con suerte se puede encontrar en <http://personales.ya.com/tardio/Zips/Td.pdf>.

¿He dicho ya que las cosas maravillosas que se hacen en España tienden a ser olvidadas, denostadas, incluso vilipendiadas en cuanto nos descuidamos? Pues eso. Nada más que añadir.

Bueno, pues el caso es que no sólo centros militares se conectaban a la red, también Universidades, Centros de Investigación, Laboratorios, Agencias... primero sólo en Norteamérica, después en el resto el mundo: en España y a principios de los ochenta, el INTA (Instituto Nacional de Técnica Aeroespacial, con sede en Torrejón de Ardoz, Madrid) estaba ya conectado y me imagino que poco a poco las conexiones se fueron extendiendo por doquier. Pero eran comunicaciones lentas, de bajo nivel, con un protocolo mínimo que, desde luego, exigía conocer de antemano dónde querías conectarte proporcionando su dirección física. Útil para investigadores, sin duda, pero inútil para el uso masivo.

Había que arreglarlo, pues. Cuando hay muchos billones de dólares en juego, las minucias tecnológicas se arreglan, y si no se pueden arreglar, se evitan, se rodean, se soslayan... o se venden como una ventaja, es cuestión de poner a los marketinianos a hacer su trabajo...

No sufráis: **os voy a ahorrar todo el rollo tecnológico**. Que, además, otros han contado mejor que yo. Supongo que sabéis que Netscape hizo el primer navegador comercialmente viable y cómo Microsoft comenzó a regalar su bastante inferior Internet Explorer, juntándolo con su Windows 95, 98 y todos los demás, y cómo esto hundió a Netscape, los juicios que siguieron... pero ésa es otra historia y será contada en otro momento.

En 1995, Sun, hasta entonces fabricante casi exclusivo de Sistemas UNIX como la magnífica serie Sun E10000 de 1997, anunció el lanzamiento de **Java**, resultado de un pequeño proyecto de Sun para crear un lenguaje inicialmente diseñado, según parece, para dotar de un entorno programático a los electrodomésticos: televisores, lavadoras, lavavajillas, cosas así. Java fue adoptado rápidamente por casi todos los competidores de Microsoft como lenguaje de programación de aplicaciones de Internet, haciéndole la competencia al Visual Basic, hasta el momento el lenguaje más usado para programar páginas web.

También hubo grandes cuitas entre los distintos paquetes de software de soporte de aplicaciones Web, con Microsoft IIS haciéndole la dura competencia a Apache, el dominador hasta entonces... pero ésa es otra historia y también será contada en otro momento.

Todos los fabricantes de máquinas UNIX se lanzaron a promocionar y vender Sistemas para dar servicio a los nuevos servidores Web, los antivirus, los cortafuegos y toda la parafernalia asociada. Y vendieron muchos sistemas y los siguen vendiendo, la verdad, aunque a otros precios.

Además, como cité hace ya algunos capítulos, en la segunda mitad de los noventa fue cuando se comenzaron a vender *en serio* ordenadores personales para los hogares. La gran mayoría de ellos llevaban ya un módem que permitía la conexión a Internet. Pero, eso sí, para conectarse a un proveedor de internet y tener acceso a la red, había, en primer lugar, que suscribirse a dicho servicio pagando una cuota mensual adicional a la que se pagaba ya a la Telefónica de turno, claro está, y, para realizar la conexión física, marcar un número de teléfono, el del proveedor, pagando la llamada, urbana o interurbana según dónde estuviera el proveedor, a la compañía telefónica, es decir, el establecimiento de llamada y los minutos que fueran, y todo ello por la línea telefónica normal, un par de hilos de cobre que llegaban a tu casa. Naturalmente, mientras navegabas no podías llamar ni recibir llamadas, dado que tu línea estaba ocupada.

Los poquísimos accesos que yo hice a internet desde casa durante esos años de fines de los noventa (tampoco había mucho que consultar o hacer, no creáis, que *la cosa* no era como es ahora, ni se le parecía), los hice sin necesidad de suscribirme a proveedor alguno, llamando a uno concreto que daba el servicio sin coste... sólo que el teléfono al que había que llamar comenzaba por “903”, un “*Número de Tarificación Adicional*” que costaba una fortuna el minuto... pero era una solución de emergencia que te podía sacar de un apuro.

Naturalmente, nadie se sabía de memoria el número de teléfono de su proveedor, así que todo el mundo lo guardaba en su “Acceso Telefónico a Redes”. Cuando lo arrancaba, el módem llamaba al número almacenado... y enseguida empezaron a aparecer virus que te cambiaban sin que tú lo supieras el número de teléfono y llamabas a un 903 o un 906... o a

un teléfono de Filipinas, que de todo había. Cuando te llegaba la factura, te quedabas sin aliento... y sin dinero. ¿Hizo algo Telefónica para perseguir a los *dialers*, que así se llamaban los programitas de marras? Pues no.

Claro, *es lógico*: una buena parte de la elevada factura que generaban era para el operador, así que no iba a perseguir una actividad que tan pingües beneficios aportaba... ¡**Aaargh, me hierva la sangre!** Porque el fraude, pues este proceder no es otra cosa que un fraude, o sea, *un robo*, sigue hoy en día con tantos y tantos timos, muchos de ellos en las televisiones nacionales o locales, de “llame Vd. a un 803 para conseguir no sé qué premio o trabajo o lo que sea...” y nunca llegas a nada, puesto que el objetivo del timador es simple y llanamente que llames y tenerte enganchado media hora al teléfono con cualquier excusa.

Ninguna telefónica de ninguna parte hace nada por impedirlo. ¿Lo hace quizá la Administración? Nones. ¿La Justicia? Va a ser que no. ¿La Policía? Tampoco.

¿Y entonces...? Pueees... ya veis, no queda otro remedio que entonar el famoso Brindis de “Marina”, la zarzuela de Emilio Arrieta: “*A pagar, a pagar y a tragar... tu pasta ya voló...*” (bueno, vale, ya sé que el original dice: “*A beber, a beber y ahogar... el grito del dolor...*”).

Ya vuelvo al tema del capítulo, ya... es que hay cosas que consiguen alterarme físicamente, y ésta es una de ellas, os ruego perdonéis a este abuelo cascarrabias.

En la segunda mitad de los noventa había ya algunas empresas que habían aportado tecnología interesante para hacer de la red un lugar más confortable. Yahoo, por ejemplo, había pasado de ser un mero directorio de sitios interesantes a un portal de éxito; Amazon había conseguido hacer realidad el hecho increíble de que una tienda online fuera capaz de generar ingresos; Altavista se convirtió en un buscador de internet de calidad; Hotmail proporcionaba correo electrónico gratuito a cualquiera (ningún otro correo de la época era gratuito); eBay empezaba ya a posicionarse en el próspero mercado de subastas de artículos entre particulares, etc.

Claro que, mientras estuviéramos constreñidos por la tecnología de cobre no podríamos aprovechar las potenciales ventajas de acceder a internet: habría que esperar necesariamente al cable. Y teníamos un problema: la penetración del cable en España era virtualmente inexistente por esa época y el hilo de cobre, el que llegaba a nuestras casas, tenía muy escasa capacidad: ya en la Carrera me aseguraron que no más de 64kbps, suficiente para una conversación o para los arcaicos módems de la época, pero completamente insuficiente para un acceso intensivo... y no sólo en la Carrera (eso era en los setenta), sino que después lo oí muchas veces.

¡*Tacháan!* Pues, de pronto, resulta que habíamos estado equivocados todo el tiempo: ¡el humilde cobre tiene unas propiedades buenísimas y admitía mucho, pero que mucho más ancho de banda! Se ve que lograrlo tan sólo era cosa de que interesara...

Nunca entendí esto muy bien: Perooo... vamos a ver: ¿Podía, o no podía? No soy Físico, ni Ingeniero de Telecomunicaciones, como creo que ha quedado demostrado página a página hasta aquí, pero para mí que esto de la *escasa capacidad del cobre* era una patraña más que nos contaban... y cuando hizo falta, o mejor, cuando se pudo vender, cuando hubo dinero contante y sonante que ganar... ¡resulta que no había tales limitaciones! En fin. Seguro que hay razones tecnológicas que lo explican, pero a mí siempre me pareció bastante raro.

Hubo entonces que aprenderse una palabreja nueva: **ADSL**. Que inicialmente



ofrecía 128 Kb, 256, luego medio mega, un mega entero... dos, cinco... Veinte! Cincuenta!! ¿Quieres televisión? ¡También! ¡Y dos huevos duros! ¿Quién da más...? Pues eso: en fin. Aunque, eso sí, una cosa es la velocidad de ADSL que te venden y otra la que tienes realmente... ¿No hay ningún organismo oficial de algo que ponga un poco de orden en esto...? Porque yo pago un ADSL buenísimo de 10 Mbs, pero no creo haber visto en la vida más de dos o dos y medio. ¿Por qué me engañan? ¿Se creen, acaso, que soy tonto...? Bueno, es que igual sí lo soy.

En los *países avanzados* y en la década de los noventa, la penetración de la banda ancha era muy superior a la que había en España, donde era lenta y cara. Así que los de aquí veíamos con cierto estupor cómo tantos y tantos consultores extranjeros, anglosajones mayormente, de pronto *se volvieron locos*. Promocionaban a voz en cuello lo ideal que era zambullirse sin ambages ni miramientos en la tecnología de internet como si fuese el nuevo Grial...

Claro, es que **ellos sí habían visto velocidades razonables y buenos contenidos**. Así que comenzaron a promocionar con fuerza inusitada la nueva tecnología de Internet. De hecho, en mi experiencia nunca he visto un despliegue de tal calibre para promocionar ninguna tecnología de ninguna clase. El motivo era evidente: la llegada masiva de la banda ancha a la población en general, centenares de millones de hogares, proporcionaría capacidad de acceso ilimitado a miles de millones de personas a los contenidos, productos y servicios ofrecidos desde cualquier punto del planeta.

Vale que el acceso a la banda ancha era por entonces bastante caro e ineficiente, al menos en España, pero... *ya bajaría de precio*, como siempre ha acontecido en las cosas tecnológicas. Y eso significaba mucho, pero *mucho* dinero para mucha gente. Por tanto, y como siempre, fabricantes, consultores y empresas suministradoras de servicios informáticos se dedicaron unánimemente a promocionar intensamente la nueva tecnología. Y esta vez, para variar, *no iban unos contra otros*, sino todos ellos remando en la misma dirección y con mucha fuerza, además.

Vale, la seguridad era casi inexistente, pero ¿a quién le importaba? Ya se solucionaría mágicamente... La tecnología específica para la Web estaba haciendo sus primeros pinitos, pero ¿a quién le importaba? Ya mejoraría más pronto que tarde... La penetración de la banda ancha era aún escasa y de poca calidad, pero ¿a quién le importaba? Pronto habría banda ancha para dar y tomar... En una palabra: ¿inconvenientes? ¡**Ninguno!**, y si los hubiera, ya sabe: contrate a su proveedor favorito de servicios profesionales, que ya se ocupará él de venderle programadores con dos meses de experiencia a precio de Jefe de Proyecto Senior... ¡Y eso que en aquellos años tan movidos estábamos todos hasta el cuello con el dichoso problema del Año 2000! (el famoso *Virus del Milenio*).

Nada nuevo bajo el sol... esa misma situación la habíamos vivido varias veces a lo largo de nuestra vida: con los mainframes; los PC's; con los Minis, el downsizing y el upsizing; con las Metodologías; los Data Warehouses y los Business Intelligences; los ERP's; los Objetos... y en otros ámbitos de la tecnología, con la telefonía móvil, los CD's, los vídeos y luego los DVD's, etc. Otra vez estábamos ante la enésima *tecnología maravillosa que nos solucionaría la vida*... nada nuevo, ya digo.

**Pero no esta vez.** Sobre todo a partir de 1998, **algo sí era diferente**. Mucho. No sólo los consultores y fabricantes de tecnología hicieron su campaña promocional... a la fiesta se sumaron consultores empresariales, gurús y analistas financieros, Bancos de Inversión, periodistas y columnistas de la prensa *salmón* (la económica, para entendernos),

y todos a coro se unieron a los tecnólogos para vendernos “**La Nueva Economía**” ligada a la tecnología.

Que yo sepa, esto no había ocurrido nunca antes. Comenzó a ser usual encontrar artículos en las cabeceras económicas más prestigiosas, como Wall Street Journal o Financial Times, alabando Internet como cambio de paradigma... empresarial.

Esta vez el mensaje no era el tradicional de “los informáticos habéis estado siempre equivocados; las cosas hay que hacerlas de forma diferente a partir de ahora y, si no, es que eres un *neanderthal*...”. Estábamos ya acostumbrados a este baile, nos sabíamos bien los pasos y no nos impresionaba mucho. No, esta vez el mensaje era mucho más potente, dirigido no al Director de Tecnología, sino *al propio Consejo de Administración* de las Compañías, verbigracia:

**“Las Compañías estáis equivocadas en vuestra manera de hacer negocios, lleváis siglos equivocados. Las cosas hay que hacerlas de otra manera, *por internet*, o si no... desapareceréis en cuatro días”.**

Se acuñó un nuevo palabro: “*brick-and-mortar*” (ladrillo y cemento), que en este caso era usado de forma despectiva: el mensaje era “**Las empresas que en dos años sigan basando su negocio en el *brick-and-mortar* (o sea, que sigan basando su negocio en la *Economía Tradicional*) serán barridas del mapa... ¡He dicho!**”.

O sea: Si es Vd. el Director de un Banco, cierre las oficinas físicas y hágalo todo *por internet*. Si es Vd. el Director de una Gran Empresa de Distribución, cierre Vd. sus Centros Comerciales y venda todo *por internet*. Si es Vd. una Agencia de Viajes, más le vale cerrar sus oficinas y vender todo, todo, *por internet*. Si es Vd., en fin, el Director de una Inmobiliaria... pues no sé, compre Vd. una buena compañía *por internet* y deshágase de los ladrillos. Todo negocio que no se haga *por internet* está muerto, no tiene futuro, es un negocio abocado sin remedio a la quiebra... **¡Y NO ME DISCUTA!!!**

Todo argumento en contra era apartado, contraatacado, pulverizado con absoluta desfachatez: Si necesitas sacar dinero de tu cuenta en el banco, vete a un cajero y si no lo hay... pues no saques dinero. Si quieres probarte la ropa antes de comprarla, apréndete de una vez cuál es tu talla y déjate de excusas. Si no sabes si el besugo que vas a comprar está fresco, pues te lo crees y punto. Si hay que hacer un visado para el Beluchistán (que como no los hace *por internet*, son unos atrasados), ¡pues no vayas a Beluchistán!, si serán eso, atrasados... Si tienes un siniestro, te lo arreglarán *por internet*, el perito lo valorará *por internet* y te pagarán *por internet*. Y las primas (las del seguro, no las hijas de tus tíos), igual: las calculas, contratas y pagas *por internet*.

Todo, todo, *por internet*. Y todo eso... ¡todavía en el Siglo XX!, con la incipiente tecnología del momento. Veinte años más tarde, muchas de estas cosas nos parecen razonables, normales, habituales... pero es que en tecnología informática, el tango *no* tiene razón: ¿Qué es eso de que veinte años no es nada? ¡Veinte años son muchísimos!

En fin, ése era el mensaje con que nos bombardeaban continuamente. ¡**Durísimo mensaje, pardiez!** Nosotros, los informáticos, estábamos razonablemente acostumbrados a los agoreros, a los oráculos y a los cantos de sirenas, era el pan nuestro de cada día y sabíamos cómo manejarlos; se ve que muchos Directores Ejecutivos de empresas grandes y chicas no lo sabían.

Se destinaron cantidades indecentes de recursos de todo tipo para comprar equipamiento *para la internet*, en contratar servicios y en fichar profesionales que supieran de qué demonios iba eso *de la internet*, porque desde luego la mayoría de las empresas no tenían ni uno, como podéis imaginar. Se comenzaron grandes proyectos para hacer no sé

qué grandes negocios diversos *por internet* y no llegar el último a la fiesta. Se invirtió una barbaridad en adquirir compañías propietarias de patentes de alguna oscura tecnología lejanamente relacionada *con internet*. Se gastaron ingentes cantidades de dinero...

Pero... **¡es que el dinero no faltaba!** Si aparecía en cualquier proyecto la palabra mágica, “*internet*”, el dinero corría a raudales, era entonces sencillísimo encontrar financiación para cualquier proyecto, por descabellado que fuera, que tuviera como objetivo hacer no sé qué tontería... siempre, eso sí, que fuera *por internet*. Decir “¡*Internet!*” abría inmediatamente las fuentes de financiación sin comprobación de ningún tipo por parte de los que se jugaban su dinero... cosa sorprendente que un servidor, ex-trabajador de un Banco, no entendía mucho.

Ésta es la parte que realmente no entiendo, cómo fue posible tanta locura colectiva de Bancos de Inversión, Fondos de Ídem, Sociedades de Capital Riesgo, Hedge Funds y demás que, según leían o veían algún documento o proyecto empresarial donde pusiera “*internet*” en algún sitio, abrían la espita de los dólares y ponían sobre el tapete financiación ilimitada. Claro que no sé de qué me extraño, viendo cómo concedían esos mismos bancos las hipotecas o los préstamos a cualquier pelagatos que nunca, nunca podría devolverlos... ¡Qué cruz!

No lo entiendo porque esto no tiene nada que ver con el conocimiento o no de la tecnología. Para que un Fondo de Capital Riesgo apoye un proyecto hace previamente un detalladísimo estudio de la viabilidad del proyecto, sus datos financieros, tecnológicos y de oportunidad de negocio, la valoración de sus líderes... y sólo los mejores y más viables reciben financiación: lo sé porque he participado en la evaluación tecnológica de alguno de estos proyectos y sé cómo se hace. Y ya digo, no entiendo las causas de tanta locura colectiva a la hora de conceder dinero a espuestas a tanto proyecto descabellado.

Bueno, **sí que la entiendo**. Tiene algo que ver con la avaricia humana y no es muy distinta de la que ha originado la *RecojoCrisis nuestra de cada día*... De hecho, llegaba tanto dinero fresco a Compañías de Inversión para que lo utilizaran en adquirir acciones o participaciones de cualquier cosa que fuera *por internet* que, literalmente, no encontraban suficientes lugares donde invertir... así que todo valía.

Es una regla básica de la economía que cuando hay mucha demanda y poca oferta, el precio sube... y subió. ¡Vaya si subió! Cuando un sesudo analista de un Banco de Inversión valoraba una Compañía que tuviera algo que ver *con internet*, el precio objetivo se multiplicaba por veinte, mientras que al riesgo de la inversión se le hacía la raíz cuadrada. Cualquier inversión sería rentable y prometía fabulosas ganancias futuras. Dinero y más dinero llegaba para invertir *en internet*...

...Y al final pasó lo que tenía que pasar. Cuando se ignoran sistemáticamente las más elementales reglas de la inversión, el riesgo y los negocios, al final suspendes el examen. Quizás apruebes algún parcial, pero en el final el batacazo está asegurado. Claro que, eso sí, por el camino se generaron jugosísimas comisiones para Entidades Financieras, Colocadores de Emisiones y Vendedores de Humo varios...

La cruda realidad es que la tecnología estaba todavía en pañales. La penetración de la banda ancha era un chiste. No había técnicos cualificados suficientes para crear, mantener y operar la multitud de aplicaciones Web que se planeaban. Y los pocos que había, buenos o malos técnicos, daba igual, pedían una fortuna por sus servicios. Ojo, que eso me parece bien; es la Ley de la Oferta y la Demanda que tantos disgustos nos ha dado a la profesión; si entonces benefició a algunos, mejor para ellos.

Prácticamente todas las empresas comenzaron, muchas de ellas a su pesar, y me

consta, proyectos *de internet*. Escribieron sus Páginas Web Corporativas. Ofrecieron en ellas sus productos y servicios. Dieron acceso a sus aplicaciones a sus clientes, notablemente los bancos, que comenzaron a desarrollar sus “Bancos en Casa”. Se permitió comprar artículos *por internet*, sobre todo las grandes compañías de distribución, grandes almacenes, tiendas de libros, discos, electrónica, etc.

Pero aquí comenzaron a encontrarse con algunos inesperados problemillas de nada... Por ejemplo: Descargar el catálogo de productos de la Compañía para su venta *por internet* y, sobre todo, mantenerlo actualizado (introducir nuevos productos y eliminar los antiguos, actualizar los precios, las promociones, condiciones especiales, etc) representa un trabajo muy importante y adicional al mantenimiento normal que antes se hacía del catálogo. Una labor que necesitaba personal adicional al que ahora lo hace... y casi nadie lo había previsto. Luego había que gestionar los pedidos, resolver las incidencias, buscar los productos y... caramba, llevárselos al cliente a su domicilio, viviera donde viviera. Resultó que no era factible todavía enviar un paquete de libros o discos o el frigorífico o lo que fuera a la casa de los clientes por la línea telefónica: había que preparar el paquete y buscar a alguien que se encargara de llevárselo al cliente a su casa, en Madrid, Barcelona... o Coria del Río, o El Ferrol, o Medinaceli...

Y todo esto ¡es **muy complicado**! De hecho, los problemas logísticos se llevaron por delante muchos proyectos preciosos en el papel, a los que la dura realidad se encargó de poner en su sitio.

Dar a los clientes acceso operativo a sus cuentas de un Banco era otro problema de gran envergadura, sobre todo debido a que, tratando con dinero, toda precaución es poca para asegurarse de que quien está haciendo la operación es realmente quien dice ser. Y la tecnología no estaba todavía a la altura. No sé yo si ahora lo está, con tanto phishing y tanta gaita, pero, desde luego, entonces no lo estaba.

A todo esto, los fabricantes de tecnología y vendedores de Servicios Profesionales vivieron sus mejores momentos. Vendían mucho, de cualquier manera y a cualquier precio. Y tenían cola: no daban abasto.

A continuación os cuento una anécdota que sirve como ejemplo de la arrogancia que derrochaban aquellas exitosas (aunque por poco tiempo) compañías de medio pelo, anécdota que tuve la oportunidad de presenciar en directo. Una de las principales compañías españolas de distribución de libros y discos tenía ya su página web, digna aunque no muy ambiciosa todavía, donde ofrecía sus productos a los pocos clientes finales que se atrevían a comprar *por internet* en aquellos años. Los responsables observaron que Amazon había incorporado un *servicio de recomendaciones* que ayudaba (o no, porque a mí jamás me ayudó nada de nada, pero eso va en gustos) a seleccionar la mercancía que te podía interesar entre tanta oferta.

La cosa venía a decir: “los clientes que se han interesado por la Quinta Sinfonía de Mahler de Leonard Bernstein, que es la que estás tú consultando en ese momento, también se han interesado por...” y aquí venía una lista de otros títulos que, supuestamente, serían de tu interés, pues otros antes que tú los habían consultado o comprado en la misma sesión que la Quinta de Mahler. Esa forma de hacer las recomendaciones era lo que esa compañía deseaba incorporar a su página web.

Nos enteramos gracias a nuestros espías que esa función la hacía un productito *de internet* que Amazon había adquirido a una compañía llamada Net Perceptions, que digo yo que ya no debe de existir, pues su web ahora lleva a un video de promoción de algo

rarísimo... Se pusieron en contacto con ellos, que resulta que tenían Delegación en Londres, y les explicaron que deseaban hacer una prueba de su *maravillosa tecnología*, para ver, en primer lugar, si simplemente se integraba bien con su web desde el punto de vista de la tecnología, y después, si se obtenía algún beneficio adicional, o sea, más venta, por hacer recomendaciones a los clientes, que no sé yo cómo pensaban ellos medir tal cosa, pero en fin. O sea, el procedimiento normal que se usa para adquirir cualquier producto importante en las empresas serias.

La respuesta que recibieron de los de Net Perceptions dejó literalmente patidifusos a los responsables de tecnología de esa gran empresa (y a mí, que sólo pasaba por allí). Más o menos, fue:

1) “El producto funciona sí o sí. Miren la web de Amazon y ya verán que sí que funciona”.

2) “Nuestra política no es probar nada. Si desean *probar* el producto, compren una licencia para pocos accesos (que costaba como 35.000 ó 40.000 euros, no era nada barato, porque... como era *para internet*...), la instalan, la prueban y, si les va bien, ya compran la licencia que necesiten. Y no, el precio no es negociable, ni devolvemos el dinero si deciden finalmente no instalar el producto”.

3) “No, no vamos a ir a España a presentar ni a demostrar nada; podemos enviar documentación desde Londres vía e-mail y el producto se descarga *por internet* una vez abonada la licencia”.

4) “No, tampoco vamos a enviar a ningún técnico para instalarlo, ni siquiera pagando (*yo creo que es que no tenían ni uno*); el producto va con detalladas instrucciones de instalación y, siguiéndolas, se instala fácil, pero fácil, fácil”.

5) “¿Traducirlo? ¿Al español? ¿Spanish? ¿Are U sure? Pues no está en nuestros planes... Ya está en inglés, ¿qué más se puede necesitar?”.

En *dos* palabras: **IM - PRESIONANTE** (¡gracias!, Jesulín de Ubrique, por aportar esta irrepetible joya al repertorio hispánico de sandeces...). Y no os creáis que era un producto de fácil instalación, no: necesitaba ciertas Bases de Datos Oracle que había que cargar y mantener, planificar y ejecutar los procesos de carga de la información y una conectividad determinada y no tan sencilla de implantar, porque era la que tenía Amazon, pero si tenías una infraestructura distinta, lo que creo recordar que era el caso, el asunto se complicaba bastante.

Creo que aquella empresa desechó finalmente el producto aquel. No sé si, con semejante política, venderían mucho y, sobre todo, si estarían sus clientes contentos... Va a ser que no, puesto que ya no existen... Pero actitudes similares se encontraban con cierta frecuencia aquellos años gloriosos.

En fin. Sigamos con nuestra historia.

El caso es que tanta presión mediática, tanto gurú evangelizando al personal dio, como era de esperar, sus frutos. ¡Vamos, que si dio sus frutos...! El panorama bursátil se dio literalmente la vuelta; las grandes compañías de toda la vida, eléctricas, petroleras, bancos, constructoras, alimentarias... de pronto se encontraron *infraponderadas* por la mayoría de analistas (claro, como basaban su negocio en el *brick-and-mortar*...) y sus cotizaciones comenzaron a ir a la baja. Muy a la baja.

Y mientras, sin embargo, las compañías que tuvieran relación en todo o en parte *con la internet* subían como la espuma, sin necesidad de que hubiera siquiera que molestarse en ponerlas en *sobreponderar*...

Por ejemplo, **CISCO**, una compañía que fundamentaba (y fundamenta) su negocio en la fabricación y venta de excelentes routers, los que instalan las empresas, no tanto los que ponemos nosotros en casa para enchufarlos a la línea telefónica, se convirtió por entonces en la Compañía más valiosa del mundo, con una capitalización bursátil de más de medio billón de dólares... billón europeo, desde luego: 500.000 millones de dólares... Más que todas las petroleras, que las manufactureras de acero, de coches, que cualquier Banco o Aseguradora del mundo... y, por supuesto, que cualquier otra compañía tecnológica.

Y eso a pesar de que el resto de tecnológicas tradicionales también tuvieron fuertes subidas... pero lo que fue sorprendente fue el comportamiento de tantas y tantas compañías que basaban su negocio *en internet* que, pese a que sus planes de negocio preveían tres, cuatro o más años de pérdidas operativas y fuertes inversiones antes de dar un solo dólar de beneficios, multiplicaron su valor inicial por diez, por veinte, por cincuenta...

theGlobe.com, por ejemplo, una compañía de la que casi nadie ha oído hablar por aquí y los que sí que oyeron en su día no recuerdan, ofrecía la creación de alguna clase de red social de no sé qué tipo *por internet*. Una especie de “proto-Facebook”, creo. En su OPV fue vendida a un precio de 9 dólares, pero en su primer día de cotización llegó a marcar un cambio de ¡97 dólares!: una subida en ese primer día de casi el 1000%, que es la mayor subida de un valor en un solo día que se recuerda. theGlobe.com, obviamente, ya no existe.

Locuras de este tipo ocurrieron en todos los países del mundo, pero me centraré en algunos casos que conozco bien, puesto que ocurrieron aquí, en España:

**Red Eléctrica de España** era y es una tranquila compañía dedicada al transporte mayorista de electricidad, es decir, era la propietaria de la mayoría de líneas de Alta Tensión del país, y de hecho lo sigue siendo, que yo sepa. Una parte de las acciones propiedad del Estado fueron ofrecidas en una OPV (Oferta Pública de Venta) al mercado, a un precio de 8 Euros la acción a mediados de 1999.

Desde entonces había llevado una lánguida existencia bursátil y su valor se había ido deslizado a la baja inexorablemente hasta alrededor de 5,60 Euros en el primer trimestre del año 2000. Entonces hizo un anuncio al mercado (un *Hecho Relevante*): iba a utilizar el terreno en el que están instaladas las líneas de Alta Tensión, propiedad suya, para instalar en los años siguientes una red de fibra óptica que permitiría mejorar el acceso y la velocidad *de internet*. Uyy... ¡había dicho la palabra mágica!

En poco más de una semana, su cotización bursátil llegó hasta más allá de los 12 Euros, más que duplicando su valor... y todo por el *anuncio* de que *tenía el proyecto de empezar a hacer algo* que quizá dentro de unos añitos valdría *para mejorar internet*. Nada tangible, nada real: ¡El doble de dinero! Así eran las cosas en el fragor de la burbuja puntocom.

**Telepizza**, por su parte, era y es una compañía que cocina y envía pizzas a domicilio, que también llevaba cotizando en Bolsa desde 1996. Como nunca repartió ni una vulgar pesetilla de dividendos, entre otras causas, también fue perdiendo valor paulatinamente. Entonces dio a conocer al mercado que había firmado un acuerdo con no me acuerdo qué compañía para repartir no sólo pizzas, sino también otros objetos, como cámaras de fotos, libros, películas, etc., que se venderían... ¡sí!, lo habéis adivinado: ¡*por internet*!

Su cotización subió bastante, no me acuerdo cuánto, en un rally importante debido al anuncio, aunque ciertamente poco le duró... por otros motivos, al poco siguió cayendo

hasta valores ínfimos hasta que fue finalmente comprada por otra empresa y retirada de Bolsa.

La misma historia, con más o menos variantes, se dio una y otra vez en España y en todo el mundo en aquellos años tumultuosos desde 1998 hasta la segunda mitad del 2000: con decir la mágica palabra ¡**abracadabra!**, presentando cualquier proyecto que tuviera algo que ver *con internet*, por descabellado que fuera, se abría la puerta de la Cueva de los 40 Ladrones y el dinero corría a raudales para esas compañías, su valor bursátil subía como la espuma, algunos se forraron... y muchos se arruinaron, después.

Un caso paradigmático en el área hispanoparlante, uno que viví de cerca, mejor dicho, que me tocó de cerca, más que a mí, *a mis finanzas*, fue el de Terra Networks. No es que yo invirtiera una peseta en esa cosa del demonio, pero había yo adquirido participaciones en un Fondo de Inversión que sí que lo hizo... y salimos trasquilados, el Fondo y, sobre todo, yo, junto con los miles de partícipes restantes, claro.

En 1996 o 1997, un par de avispados emprendedores españoles fundan un portal en español, de nombre Olé.com. La dirección de Olé.com lleva hoy a Terra, así que no la busquéis... Este portal, creado teniendo como inspiración, más bien en menos que en más, lo que Yahoo estaba haciendo en EEUU, tuvo cierto éxito, convirtiéndose en uno de los portales más visitados en el incipiente nacimiento *de internet* en España.

En 1999, tras una serie de bien diseñadas *operaciones de ingeniería financiero-contable* que dejaron pingües comisiones en según qué bolsillos, el portal fue finalmente adquirido por Telefónica, que lo usó de base para crear y lanzar **Terra**, su portal de acceso a internet.

Telefónica pagó unos 2.000 millones de pesetas para hacerse con el portal Olé.com (unos 12 millones de Euros: ahora parece una fruslería, pero entonces era muchísimo dinero), invirtió muchísimos miles de millones más para adquirir otros portales hispanoamericanos, sólo en 1999 la mareante cantidad de 100.000 millones de pesetas, o sea, unos 600 millones de Euros más (para los que no lo sepáis o no os acordéis, un Euro eran 166,386 pesetas, o lo que viene a ser lo mismo, 6 Euros son casi exactamente mil pesetas) y poder realizar por fin el lanzamiento a Bolsa de la compañía resultante.

En la ilustración siguiente tenemos una captura antigua del portal de Olé.com, para que os hagáis una idea de qué era y qué ofrecía.



La historia completa de Olé.com durante aquellos años curiosos la podéis encontrar, si es que no se ha retirado, en esta página web:

[http://www.taringa.net/posts/info/1260132/Una-web-en-el-recuerdo:-Ol%C3%A9\\_com.html](http://www.taringa.net/posts/info/1260132/Una-web-en-el-recuerdo:-Ol%C3%A9_com.html), que contiene informaciones escritas precisamente a finales de 1999, cuando se estaba cocinando todo el asunto y donde, además, podréis haceros una idea de qué cosas pasaban, qué cosas se hacían y cómo algunos aprovechados y listillos se forraron.

El lanzamiento final de Terra al Mercado se produjo en noviembre de 1999 con un precio inicial de salida de 11,81 Euros. Pero... Terra era una compañía *de internet*, que haría supuestos negocios *por internet*, así que las valoraciones de los analistas financieros eran unánimes: *Sobreponderar, Sobrecomprar, Pillar todo lo que puedas antes de que se acabe...* y en su primera sesión de cotización, Terra vio cómo su valor bursátil se triplicaba, hasta más de 36 Euros por acción... y eso no es todo: el rally continuó y esta ya triplicada cotización se casi quintuplicó en los siguientes tres meses: en febrero de 2000 su valor llegó a ser de 157 Euros... y la mayoría de Analistas Bursátiles continuaban inflando sus *valoraciones objetivas*, ésas que, de puro *objetivas*, tanto me gustan a mí. Casi todos pronosticaban que alcanzaría los 250, 300... ¡500! Euros a corto y medio plazo...

En ese momento Terra era la cuarta o quinta compañía más valiosa del mercado español, con un valor bursátil de casi 6 billones de pesetas, unos 36.000 millones de Euros, nada menos, y valía más que la mayor petrolera española (Repsol), la mayor compañía eléctrica (Iberdrola) y el tercer Banco (Banco Popular)... *juntos*. A todo esto, como es natural, Terra perdía anualmente ingentes cantidades de dinero, dado que en su Plan de Negocio preveía ganar dinero sólo a partir del tercer año de funcionamiento... no obstante lo cual el valor de su acción subía y subía... y la empresa perdía y perdía más y más dinero. Raro, ¿no?

A los analistas financieros y a las cuentas de resultados de sus Bancos de Inversión esto quizá les hacía felices, pero a mí siempre me pareció una locura. Para mi anticuado punto de vista, el valor de una empresa debe estar relacionado con las expectativas reales de negocio, de generar ingresos y, sobre todo *de ganar dinero* en todos los plazos, no sólo a



quince años vista... y Terra, en 1999 y 2000, no era ni más ni menos que una máquina de perder dinero y lo seguiría siendo durante bastante tiempo aún.

¿Cuál sería el motivo por el que tantísima gente, tanto Fondo de Inversión y tantos particulares decidieran asociarse a semejante proyecto, poniendo su dinero en juego? Pensemos juntos, a ver si llegamos a algo... Y seamos generosos... muy generosos.

Imaginemos que Terra hubiera sido capaz de intermediar el 80% de todo el comercio electrónico generado en los países de habla hispana en los años siguientes (mucho decir, pero bueno). El comercio electrónico estaba por entonces en pañales, dijeran lo que dijeren los Analistas, pero supongamos que creciera a un espectacular 50% anual (que va a ser que no... pero bueno). Y supongamos que Terra consiguiera una fabulosa comisión del 3% (que también va a ser que no, pero bueno, también) por intermediar en tanta transacción comercial. Y, de paso, supongamos, además, que a nadie se le ocurre montar algo similar para hacerle la competencia en su negocio (cosa completamente increíble, porque el modelo de Terra era muy sencillo de imitar si los ingresos fueran *tan* jugosos, pero bueno, otra vez). Supongamos todo eso...

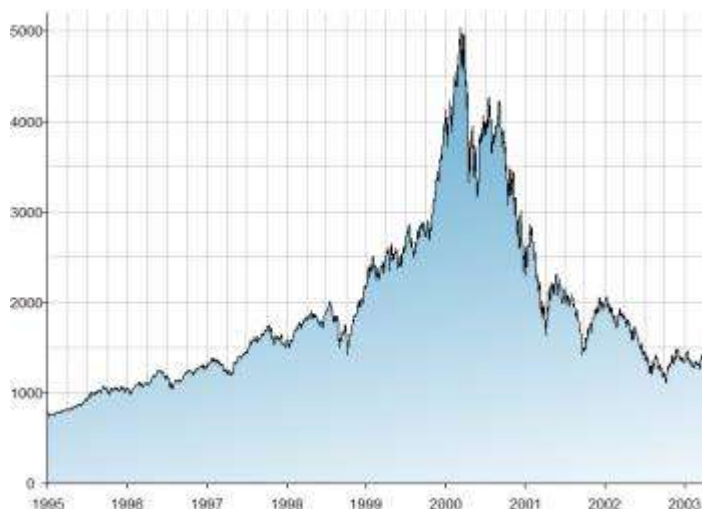
Bien, pues entonces necesitaría Terra unos **ochenta años de beneficios sostenidos** para poder pagar la compañía. Eso es lo que se llama PER, de *Price-Earning Ratio*, que es uno de los indicadores básicos que todo el mundo mira para determinar la conveniencia de la adquisición de títulos de una compañía. Un PER de 80 es malo, *muy* malo. Para que os hagáis una idea, en el año 2009 el PER de Telefónica, una de las compañías más sólidas del momento en Europa, con una política generosa de pago de dividendos y un enorme potencial de crecimiento, era de alrededor de 9 ó 10. Después de ese año, el PER era aún más bajo, como de 6 ó 7... Un PER de 80 no es que sea malo: es una catástrofe. Pero es que, además, en el año 2000 el PER de Terra era negativo, dado que, encima, no ganaba, sino que *perdía* dinero...

Me estoy poniendo un poquitín pesado, ¿no?, así que sólo diré que, de pronto, en algún momento de fines del año 2000 o principios del 2001, volvió la cordura al mercado. Las empresas, los analistas financieros, los de riesgos... recordaron de repente cuál era su trabajo, desecharon las previsiones infladas e inverosímiles que habían sido lugar común en aquellos años... y las aguas volvieron a su cauce. Igual os suena de algo a alguna incierta *Crisis de las Hipotecas Basura*, que es lo mismo).

La cotización de Terra comenzó a caer con fuerza. Todavía muchos analistas decían que había que aprovechar el momento de corrección para comprar barato, ¡hombre, por favor!, pues el rebote era inminente... pues no hubo rebote. En 2003, Telefónica lanzó una OPA para recomprar las acciones de Terra, a... ¡5,25 Euros!... tres años antes, las había vendido a más de 11. Por fin, tras marcar mínimos en 2,75 Euros, en 2005 por fin fue finalmente absorbida por Telefónica, pagando algo más de 3 Euros por acción.

Si lo deseáis, se puede ampliar toda esta historia en la página web: <http://diariored.com/blog/ana/archivo/000424.php>. Buen negocio para algunos, como veis, pero malo, muy malo para muchos.

Y lo mismo que pasaba en España pasó en todo el resto del mundo. Las empresas de internet comenzaron a caer rapidísimamente y, después, a quebrar, a desaparecer, a absorberse unas a otras, y esas *unas* fueron a su vez absorbidas por *otras* otras... Podéis observar la Montaña Rusa que fue la cotización del índice Nasdaq aquellos años turbulentos en la ilustración siguiente.



La burbuja explotó, claro. El dinero se acabó. Las otrora boyantes empresas *de internet* se dieron unos batacazos de impresión. Y entonces ningún gobierno salió en su ayuda, como ahora sí ocurre... así que cerraron, quebraron, los inversores perdieron su dinero y el capitalismo escribió otra página memorable a mejor gloria de Adam Smith.

Tantas inversiones desenfrenadas en tecnología que tuviera que ver, aun remotamente, *con eso de internet*, acabaron por volverse contra quien las hizo. Muchos se arruinaron... Y los informáticos, tras una época de vino y rosas como no habían visto los días... pues sufrimos también las consecuencias. Despidos, congelaciones de sueldo, más trabajo y menos salario... la tortilla dio la vuelta y la sartén volvió a tenerla por el mango quien tradicionalmente la había tenido: las empresas. ¡Bienvenidos al Siglo XXI, colegas!

Y para rematar la faena, y por si todo esto fuera poco, en septiembre de 2001 una pandilla de locos asesinos suicidas decidieron que no tenían mejor cosa que hacer que practicar el tiro al blanco usando las neoyorquinas Torres Gemelas como diana y aviones comerciales como proyectiles... y el mundo cambió.

¿Qué queda de todo aquello? Pues que algunas compañías fueron viables, claro que sí. La tecnología es ahora, obviamente, mucho mejor, más segura (bueno, esto no está tan claro), está más extendida... Las empresas escriben sus nuevas aplicaciones para la intranet que todo el mundo tiene ya: permite tener una aplicación centralizada pero, en el puesto del usuario, una interfaz visual con muchos colorines, nada de las viejas pantallas de fósforo de 24 líneas y 80 columnas.

En una palabra: **La tecnología en sí se ha impuesto definitivamente**, sin duda alguna. El método normal de acceso a las nuevas aplicaciones en todas las compañías que se precien es vía **intranet**, que no deja de ser como la internet, pero a escala empresarial. Hay banda ancha a cascoporro y precios *no tan caros* como al principio ocurría...

Los PC's son muchísimo más potentes. Hay tablets y smartphones, phablets y qué sé yo cuántos aparatos nuevos. Los Sistemas Operativos se cuelgan menos. El spam nos tiene a todos fritos... En fin, es muy bien conocido el estado actual de internet, así que ya me callo.

Y en cuanto al mercado... pues el mercado "*se consolidó*", o sea, muchas compañías desaparecieron; otras fueron adquiridas por otras compañías más grandes o ambiciosas... y las mejores tuvieron éxito, consolidaron su modelo de negocio, explotaron

ciertos nichos de negocio, incluso los inventaron, y ganaron y siguen ganando dinero. Amazon, e-Bay, Verisign, Facebook, etc. son compañías con un sólido modelo de negocio y entre ellas se encuentran algunas de las más sólidas compañías de la actualidad, como son Google o Apple.

Todas ellas basan su modelo de negocio en un solo concepto, un *viejo y manido concepto*, pero que sigue siendo la base de los negocios: **la Confianza**. Compras en Amazon porque *te fías de ellos*, y lo mismo ocurre con las mejores “Tiendas en Casa” españolas: se han ganado su credibilidad a pulso, demostrándolo día a día. Pujas en las subastas de eBay porque *te fías de ellos* y sabes que cumplen lo que prometen. Usas Google porque encuentras lo que buscas y, si eres empresario, contratas tu publicidad con Google porque sabes que son honrados y no te cobrarán por clics no realizados...

**Honradez, Confianza, Credibilidad:** la base para el comercio desde los tiempos de los fenicios... **¿por qué se nos olvidará tantísimas veces?** Pero ésa es otra historia y no sé si alguien sabrá contarla en algún otro momento...

## Epílogo

Hemos visto cómo empezaba el milenio con su particular crisis, para variar. Luego la economía creció mucho, luego dejó de crecer, luego se sumergió en las profundidades... y yo... yo prácticamente dejé de tener contacto diario con mi profesión. Me convertí en un Usuario más, en un *Usuario Final* de esos que sufre cotidianamente a sus compañeros (y, sin embargo, amigos) del Departamento de Informática. Y ya no tengo nada excesivamente interesante que contar de esos años. Muchas horas, mucho stress... y poca informática, como no sean algunas Hojas Electrónicas y muchos, pero muchos informes...

Mi historia dejó de ser interesante, si es que alguna vez lo ha sido, así que ha llegado el momento de echar el cierre. A lo largo de todos estos capítulos he repasado la Historia de la Informática que yo he conocido. Que seguro que no es igual a cualquier otra Historia que cualquiera de vosotros pudiera contar... ni siquiera a la Historia *Oficial*, si es que tal cosa existe. Pero, si existiera, tampoco os fiéis mucho de ella: desde Suetonio y sus *Doce Césares*, alabar a los que tienen el poder dejándolos en buen lugar y denigrar a los que cayeron por el camino es práctica común... al fin y a la postre, los historiadores también tienen que comer y eso resulta más fácil si se está en sintonía con los poderosos.

Y en cuanto al título... mi hija asegura que incluso el propio nombre que elegí (“**Memorias de un Viejo Informático**”) está mal puesto; en su lugar, ella opina que sería mucho más adecuado “Memorias de un Informático *Viejo*”... amor filial, está claro.

Muchas confidencias he contado... cosas que posiblemente no os cuente nadie más, dado que, como habéis podido comprobar, además de informático y viejo, uno es también un poco filósofo... Y lógicamente me he dejado mucho en el tintero y mucho habréis echado en falta; aun siendo culo de mal asiento como soy, obviamente no he estado en todas las tecnologías que en la Informática han sido.

AS/400, Netware, UNIX, Linux, los Macintosh de Apple, la tecnología Cliente/Servidor, Java, etc., han aparecido marginalmente en los capítulos correspondientes, pero nunca he trabajado en serio con estos adminículos... y de lo que no sé procuro no escribir.

40 años de profesión dan para mucho. Mucho trabajo, muchas anécdotas, muchas noches en blanco, muchas empresas distintas, muchos compañeros, muchos clientes, muchas actividades diferentes, desde el desarrollo de software puro y duro a la venta de tecnología y de servicios, pasando por la consultoría...

Todo eso me ha dado una visión amplia, muy personal y sí, un poco *atravesada* de la tecnología y para qué sirve... Visión que, dados mis años, me puedo dar el lujo de exponer aquí sin necesidad de ser *políticamente correcto*. Odio que por ser *políticamente correcto* se tomen decisiones equivocadas o, peor, injustas. **Odio la corrección política si va en contra de la verdad**... y ¡hay tantas veces que se omite, se disfraza, se oculta la verdad para regalar los oídos de alguien!

He contado, o esa sensación me ha dado, una y otra vez las mismas cosas... Las mismas motivaciones, los mismos errores, las mismas situaciones, los mismos eventos aplicados a momentos diferentes y a tecnologías distintas... ¿Por qué tendré yo esa sensación continua de *déjà vu*?, viviendo una y otra vez las mismas situaciones, cayendo una y otra vez en los mismos errores vestidos con diferentes disfraces cada vez: que si el

Unix debe sustituir al mainframe, que si las Bases de Datos Relacionales, a las Jerárquicas, que si el OS/2 al MS/DOS, pero el Windows al OS/2, pero el Linux al Windows y el Mac, a todos... y así *ad infinitum*, como si me moviera siempre en espiral, repitiendo los mismos ciclos una y otra vez como si fuera un pobre *pajeño* de “*La Paja en el Ojo de Dios*”, de Jerry Pournelle y Larry Niven.

Estos capítulos se han publicado en [www.eltamiz.com/elcedazo](http://www.eltamiz.com/elcedazo) como entradas, ya lo sabéis. Y han sido artículos largos, muy largos. Muchísimo más de lo estándar y usual en internet y mucho más de lo que yo pretendía originalmente, pero cuando me ponía a escribir y contaba esto, siempre me parecía que debía contar también aquello y luego aquello de más allá, cómo no... y así, casi por voluntad propia, han salido artículos de cinco, seis, siete mil palabras. Y a pesar de ello, han tenido una acogida extraordinaria, lo que agradezco en el alma.

Hay una última reflexión que quiero compartir con vosotros, amables lectores: a lo largo de tantos capítulos dedicados a tecnologías diversas han aparecido decenas de nombres de ingenieros, científicos y emprendedores que han contribuido con su trabajo y con sus ideas a que nuestra querida profesión sea, para bien o para mal, lo que hoy en día es. Hay holandeses, suizos, alemanes, franceses, italianos, indios, muchos ingleses y muchísimos estadounidenses... pero **ningún español**... Ni mexicano, ni peruano, ni argentino. Ni venezolano, chileno, ni uruguayo, ni ningún Martínez o Pérez californiano descendiente en tercera generación de un inmigrante hispanoamericano.

Se ve que la informática es cosa de *ellos*. Ya lo decía irónicamente D. Miguel de Unamuno a principios del Siglo XX: “¡**Que inventen ellos!**”... y lo malo es que, colectivamente, nos lo hemos creído.

Y es injusto. **Muy injusto**. He conocido y conozco a excelentes técnicos españoles e hispanoamericanos que no tienen nada que envidiar ni en capacidad ni en creatividad a ningún gurú de habla anglosajona. Si es caso, nuestro inglés es ligeramente peor... aunque ¡muchísimo mejor que su español! **Somos realmente buenos diseñando y escribiendo software**. Pero, eso sí, **somos muy malos comercializándolo** o simplemente dándolo a conocer... Un amigo mío siempre dice que esto es algo que tiene que ver con la idiosincrasia española, acuñada a lo largo de milenios de invasiones provenientes de los cuatro puntos cardinales, que nos ha enseñado, a golpes, a vivir mucho más al día que en otras latitudes más frías. Y lo ilustra mi amigo con un ejemplo:

¿Qué hace el *estadounidense medio* si le tocan cuarenta millones de dólares en la Lotería? Funda una empresa o, mejor aún, compra la empresa en la que trabaja hasta el momento, cambia su visión, su misión, contrata a los mejores especialistas, se mata a trabajar y se forra... O se arruina, vuelve a buscar trabajo y sigue como antes, pero escribe un libro contando su historia y se forra...

Y ¿qué hace el *español medio* si le tocan cuarenta millones de euros en la Lotería? Supongo que el ejemplo vale igual para el *argentino medio*, el *chileno medio* o el *mexicano medio*... En una palabra, ¿qué haríais *vosotros*, queridos lectores, en ese más que improbable caso?

Pensadlo, pensadlo bien un momentito...

Bien. A ver si lo adivino: Entráis en el despacho de vuestro jefe, le cantáis las cuarenta con o sin razón, dais un portazo y os largáis, muy dignos, diciendo: “¡Y el finiquito, te lo guardas muy bien guardado *donde tú y yo sabemos!*”. Os compráis un cochazo nuevo, una casa nueva, os hacéis un Crucero alrededor del mundo... y metéis el

dinero sobrante en algún Banco del que os fiéis y os dedicáis a vivir de las rentas, sin dar un palo al agua, todo lo que os quede de existencia. ¿He acertado, por ventura? A lo mejor ésa es precisamente la diferencia y explica por qué hay tan pocos nombres acabados en “-ez” entre los pioneros informáticos... o de cualquier otra tecnología moderna.

Es una pena... pero yo ya no voy a hacer nada por remediarlo. Mi turno ha pasado. Ahora es el vuestro, jóvenes lectores: **¡No hagáis las mismas estupideces que hemos hecho vuestros mayores!** Ojalá podáis.

Bien. Al fin, aquí terminan las Memorias de este Viejo Informático.

A algunos, los de mi quinta, os habrán quizá recordado tiempos pasados y muchas veces, mejores, como yo los he recordado a la vez que escribía sobre ellos.

A otros, más jóvenes, os habrá dado una información que sé por experiencia que es realmente difícil encontrar entre el marasmo de información, desinformación y demás que hoy se encuentra por doquier y quizás os haya contado algunas cosas nuevas e inauditas...

Y a los que no sabéis nada de informática... seguiréis sin saber nada, pero igual habéis pasado un buen rato leyendo sobre las aventuras y desventuras de *un pobre programador del Jurásico venido a más...*

En cuanto a mí, **declaro solemnemente que ha sido un placer escribirlas**, que me ha hecho recordar momentos muy agradables de mi vida, divertidas anécdotas, instantes mágicos y, por qué no, también amargos... es lo que tiene la vida: que más vale vivirla con todas sus consecuencias que dejarla pasar.

En consecuencia, amigos, no me queda más que pedirlos...

**DISFRUTAD DE LA VIDA, mientras podáis.**

*Macluskey*

Madrid, 2009 – 2014

